LA-UR-21-27928
Approved for public release; distribution is unlimited.

Title: Optimizing and Extending the Functionality of EXARL for Scalable Reinforcement Learning

Author(s): Chenna, Sai Prabhakarrao
Cosburn, Katherine Saara Birgitte
Ezeobi, Uchenna Mark
Moraru, Maxim

Intended for: CoDesign Summer School Exit talk

Issued: 2021-08-09

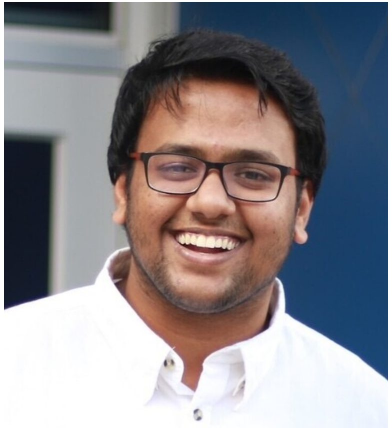
Los Alamos
NATIONAL LABORATORY

Optimizing and Extending the Functionality of EXARL for Scalable Reinforcement Learning

Sai Chenna, Katherine Cosburn, Uchenna Ezeobi, Maxim Moraru

August 5, 2021

Co-Design Summer School Student Team



Sai Chenna

PhD Student, Computer Eng.
University of Florida

Katherine Cosburn

PhD Student, Physics
University of New Mexico

Uchenna Ezeobi

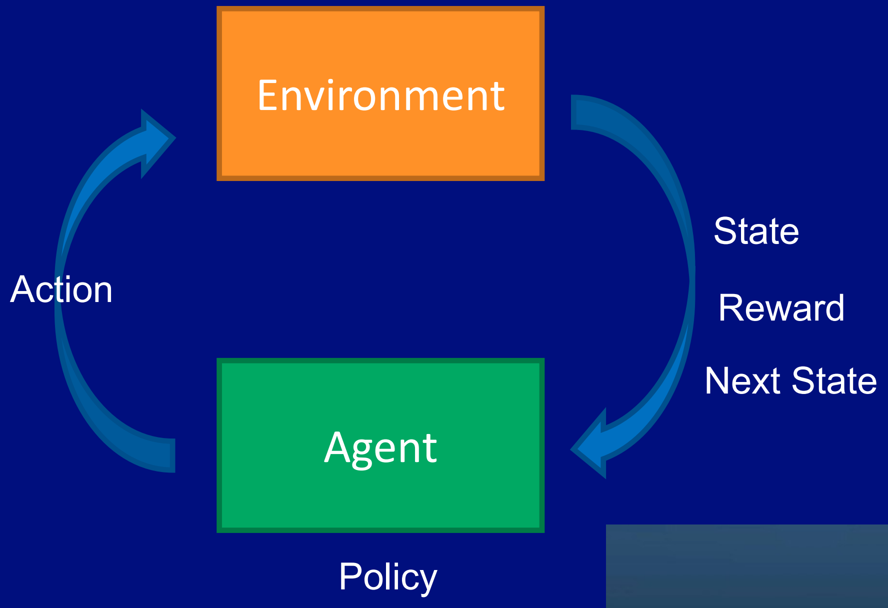
PhD Student, Computer Sci.
University of Colorado,
Colorado Springs

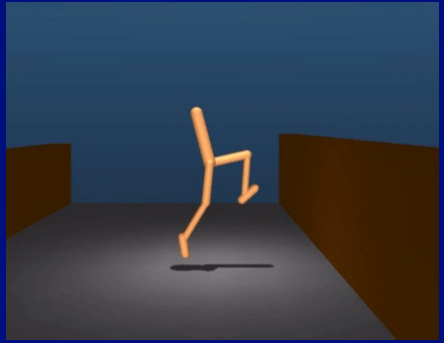
Maxim Moraru

PhD Student, Computer Eng.
University of Reims

Introduction to Reinforcement Learning

- A subset of machine learning wherein an **agent** interacts and learns from its **environment** over time.
- The agent receives a **state** from its environment and selects an **action** according to its **policy** (a mapping from states to actions).
- The agent then receives the **next state** and a scalar **reward** from the environment.
- **Goal is to achieve the maximum amount of reward over time.**

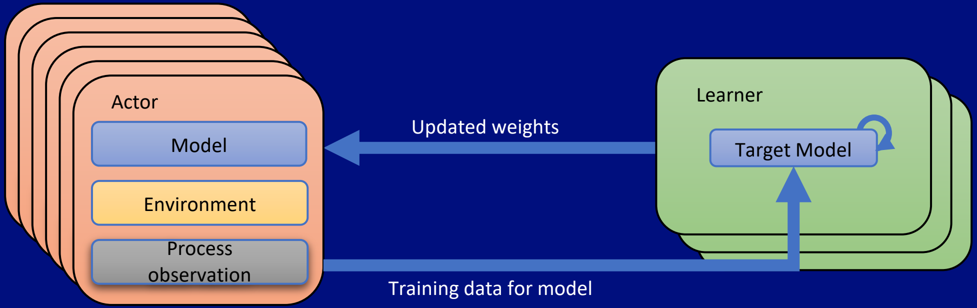




https://mofanpy.com/static/results/reinforcement-learning/6-4-demo_google2.gif

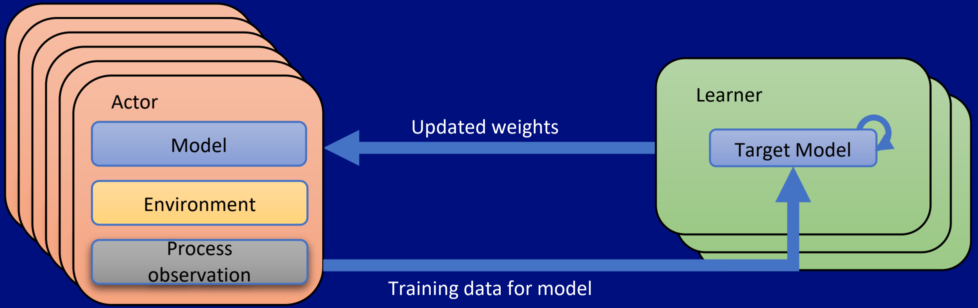
Introduction to EXARL

Easily eXtendable Architecture
for Reinforcement Learning



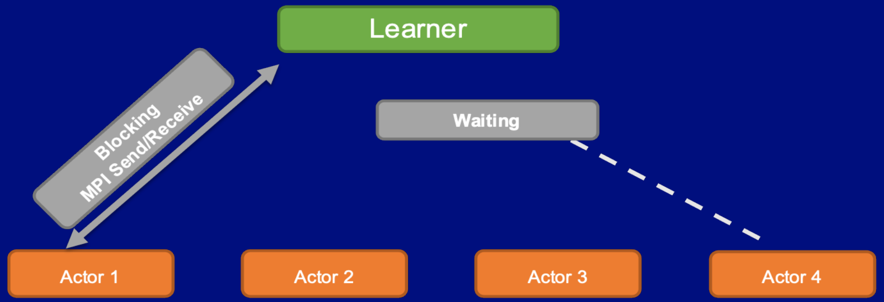
- EXARL is a **scalable reinforcement learning framework for scientific environments.**
  - Originally developed as part of ExaLearn through Exascale Computing Project (ECP)
- Why **scalable**?
  - Scientific environments are *complex* and often take a long time to run, even while running in parallel
  - Ability to run on multiple nodes reduces this time
- The **goal of EXARL** is to make prototyping and reproducing scientific RL studies easier by…
  - Providing a framework of agents, environments, workflows that are easy to add and implement
  - Having a user-friendly front-end interface (written in python)
  - Supporting different hardware and software infrastructures

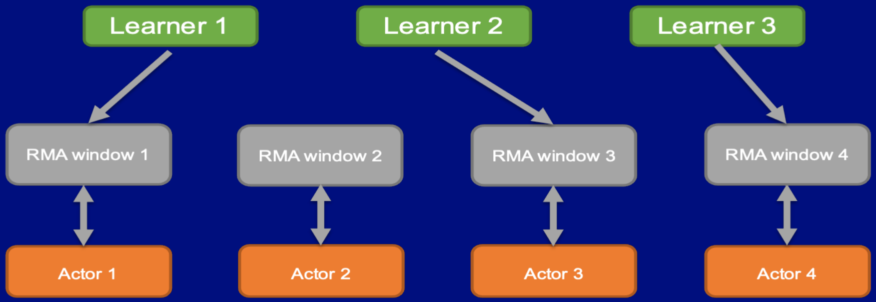
EXARL architecture



- **Agent** is decomposed into **Actor** and **Learner**
- **Actor:**
  - Gets the model/policy from the learner
  - Interacts with the environment by taking action based on the model/policy
  - Receives the trajectories (state,action,next-state,reward) from the environment
  - Sends the batched trajectories to the learner to update the policy
- **Learner:**
  - Receives trajectories from actor
  - Updates the model/policy based on the new data
  - Sends the updated policy to the actor
- **EXARL** provides a **scalable framework** for reinforcement learning
  - **Multiple actors & environments** – to accelerate learning process
  - **Multiple learners** – to accelerate policy update process

EXARL Implementation: Asynchronous vs RMA workflow





Asynchronous workflow:

- Blocking two-sided (MPI-Send/Recv) communication b/w actors and learners
- **Pros:** Actors receive recently updated policies
- **Cons:** Only has single-learner implementation, poor scalability, low policy update frequency

RMA asynchronous workflow:

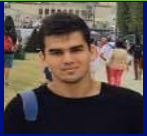
- One-sided(MPI-RMA) communication b/w actors and learners
- **Pros:** Supports multiple learners, high policy update frequency – decoupled actors and learners
- **Cons:** Policy and experience lag between actors and learners

Overview

Main goal of the Co-Design Summer School 2021 is to provide **algorithmic improvements to EXARL framework**. This is in the form of:

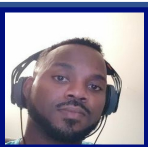
Improving Performance

- Scaling asynchronous workflow to multiple learners
- Improve scalability/execution time of multi-learner workflows
- Accelerate Deep Q-Network (DQN) data generation pipeline



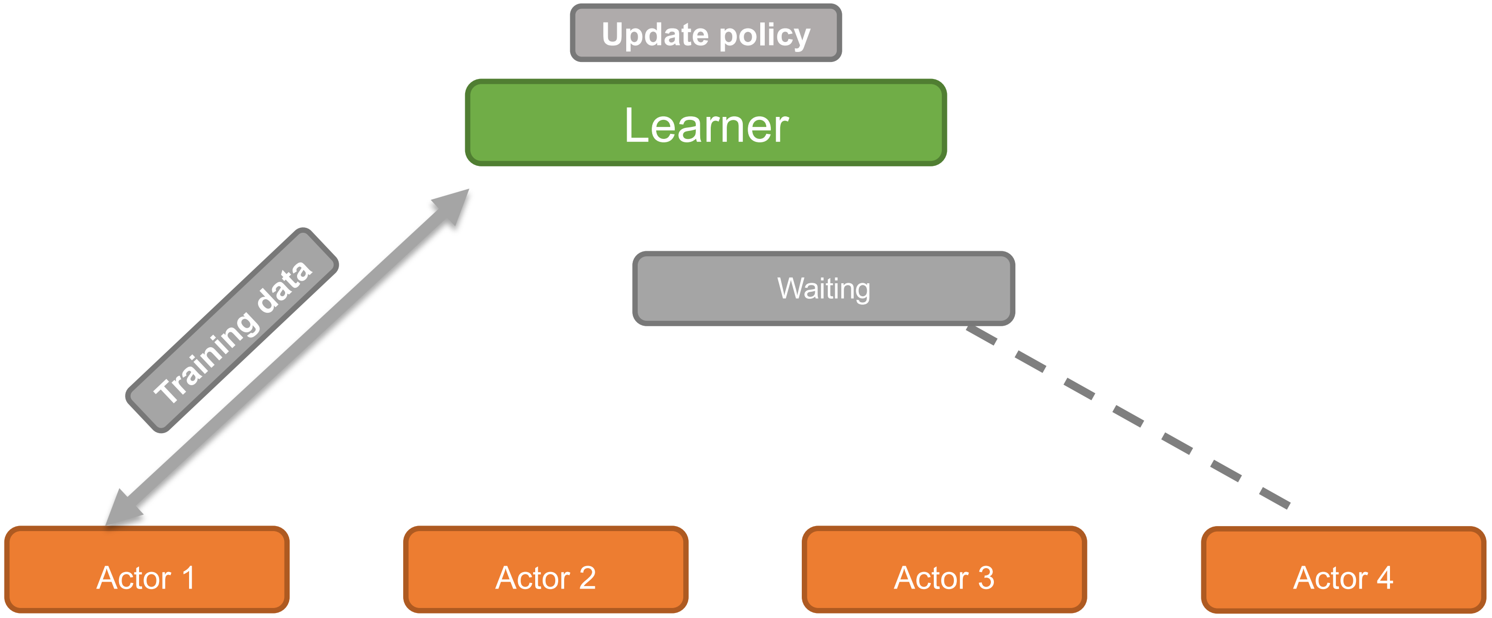
Adding Functionalities

- New agents: Advantage Actor Critic (A2C/A3C), Twin Delayed Deep Deterministic Policy Gradient (TD3)
- V-trace algorithm
- Priority Experience Replay



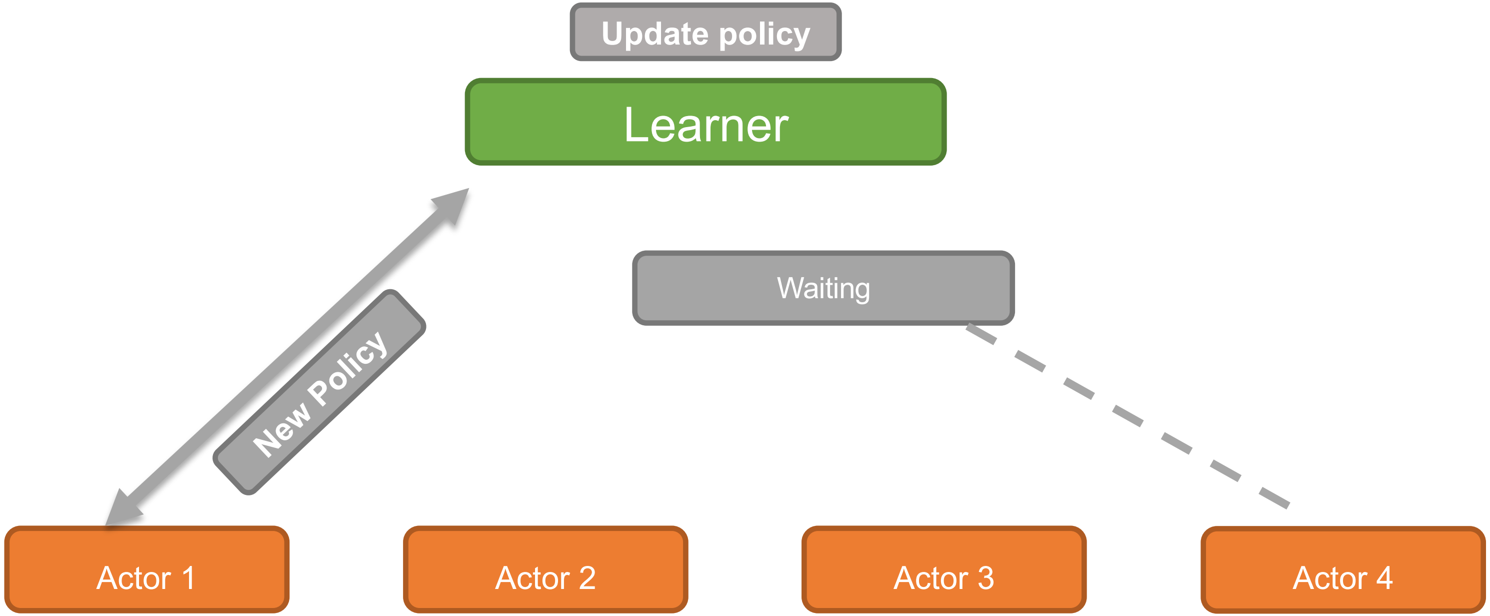
Building Multi-learner Asynchronous Workflow

- **Current approach**: Asynchronous workflow only supports single-learner



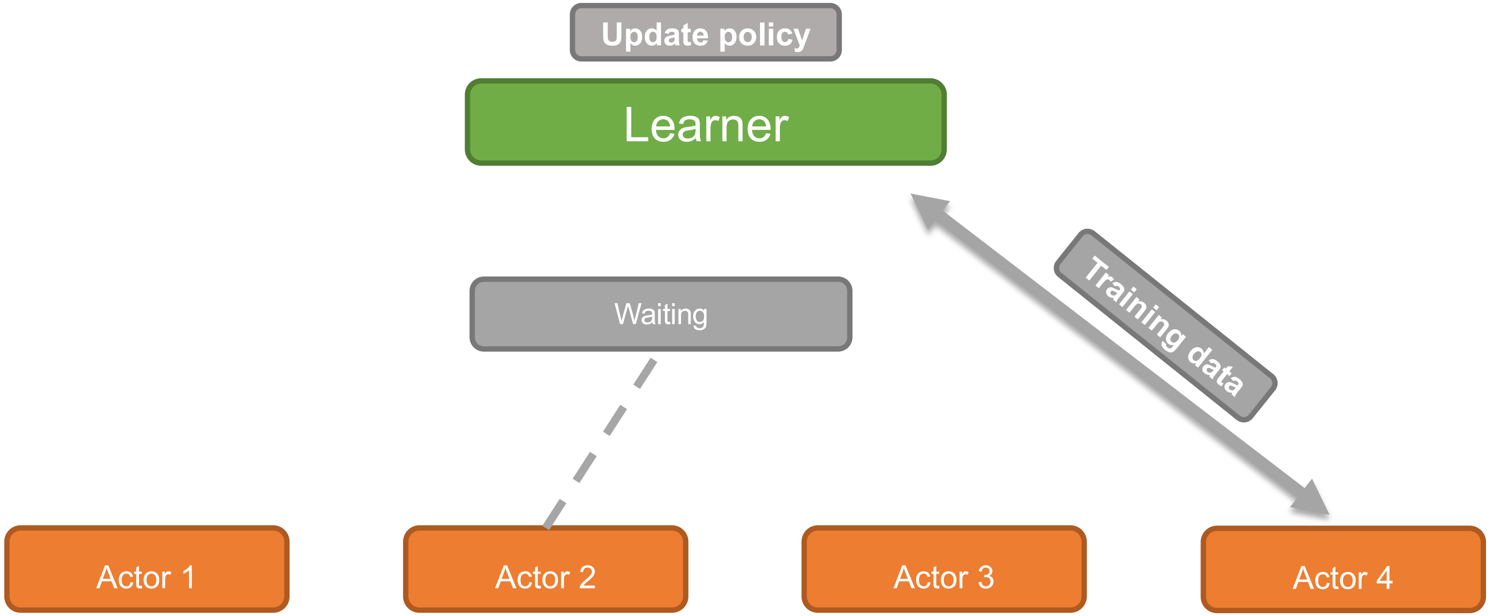
Building Multi-learner Asynchronous Workflow

- **Current approach**: Asynchronous workflow only supports single-learner



Building Multi-learner Asynchronous Workflow

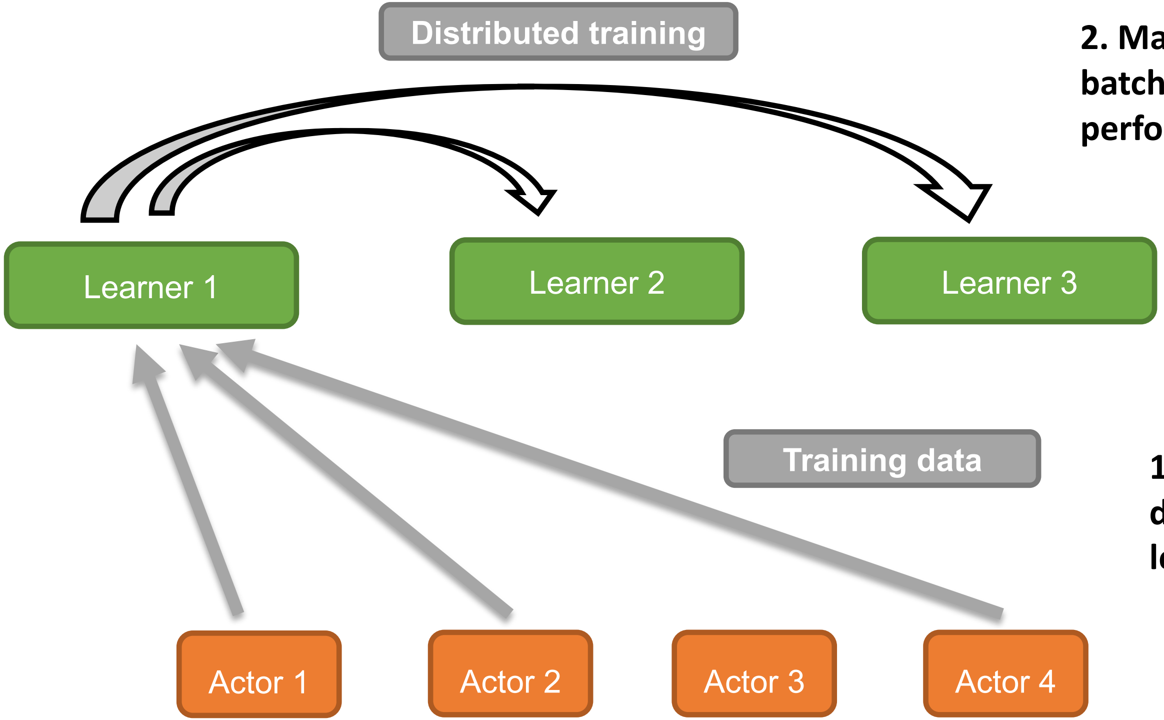
- **Current approach**: Asynchronous workflow only supports single-learner



- **Limitation**:
  - Slow training time in case of multiple actors and/or fast environments

Multi-learner Asynchronous Workflow

- **Update**: Implemented multi-learner asynchronous workflow

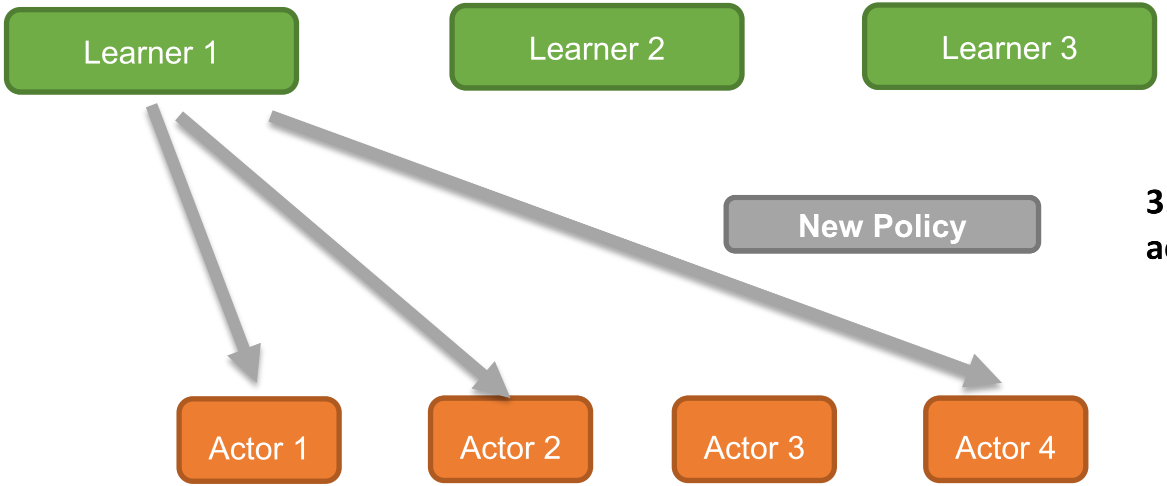


2. Master learner distributes batches to remaining learner to perform distributed training

1. All actors send the training data/trajectories to the master learner (Master-Worker)

Multi-learner Asynchronous Workflow

- **Update**: Implemented multi-learner asynchronous workflow
- **Pros:** Faster training
- **Cons:** Low policy update frequency

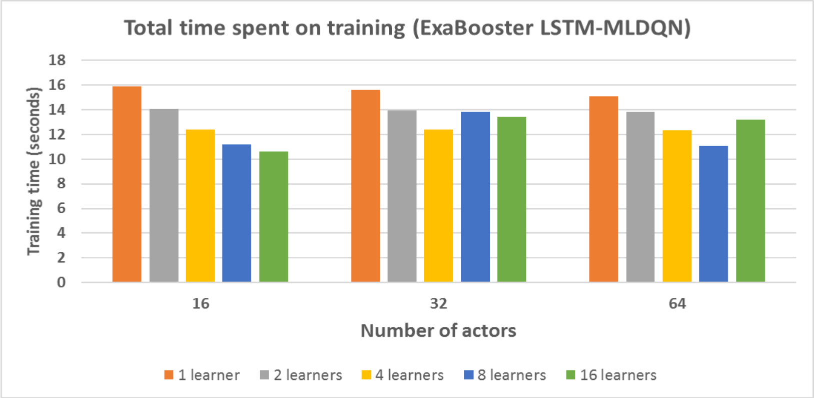


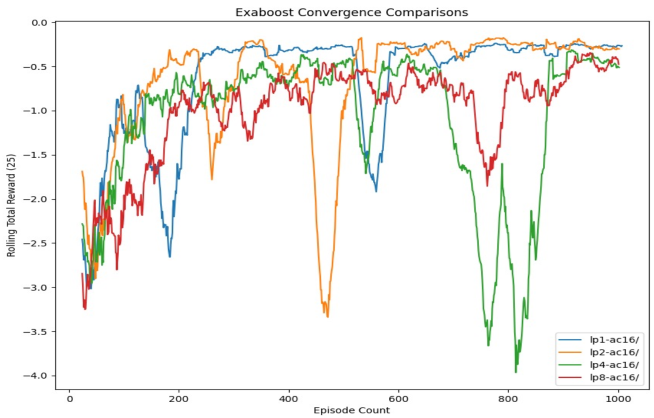
3. Updated Policy is sent to the actors

Multi-learner Asynchronous Workflow: Results

- **Experimental Setup**:
  - System: Darwin | Node: Intel Broadwell (36 cores)
  - Partition: Scaling
  - Environment: ExaBooster
  - Episode count: 1000 | Step count: 200

- **Observations:**
  - **Reduced** training time with multiple learners
  - **Poor** convergence with multiple learners
    - Low policy update frequency
  - Multiple learner workflow more suitable for **on-policy** agents
    - Policy update after every episode

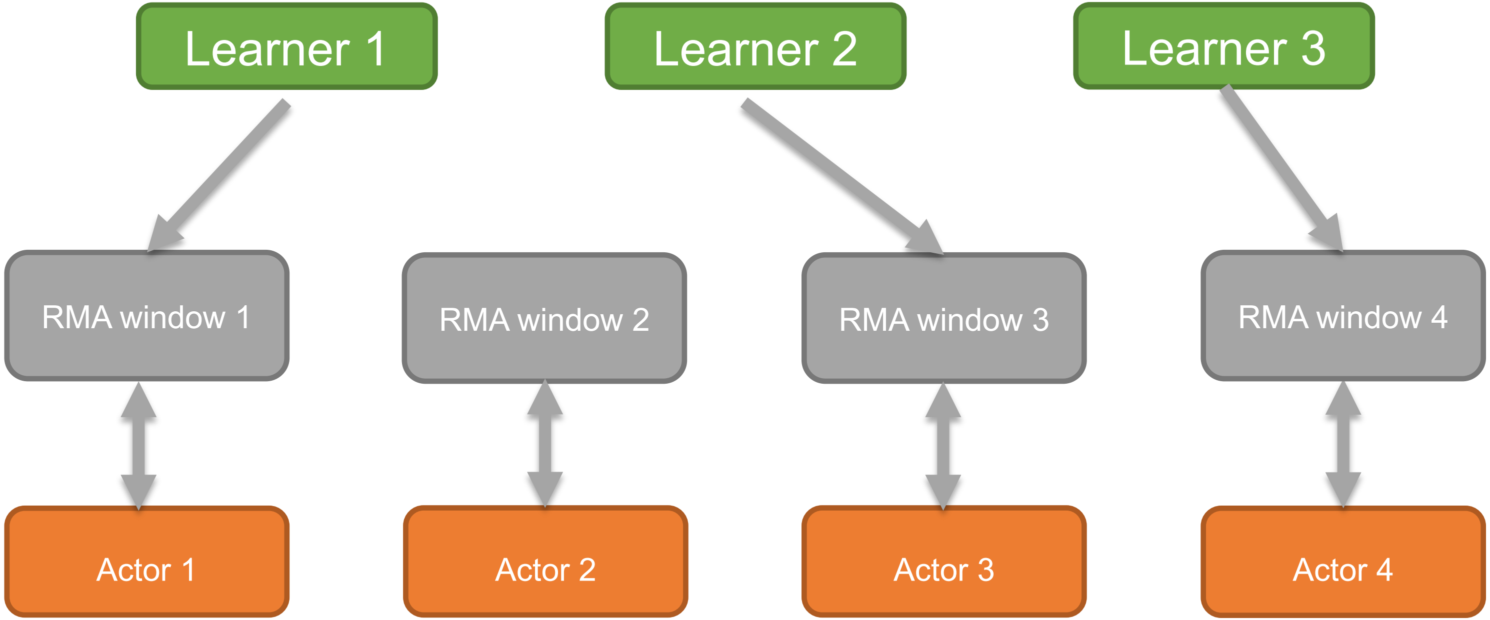




Multi-learner RMA Workflow: Window-Selection Policy

In Multi-learner RMA workflow, learner gets the training data from the actor's RMA window

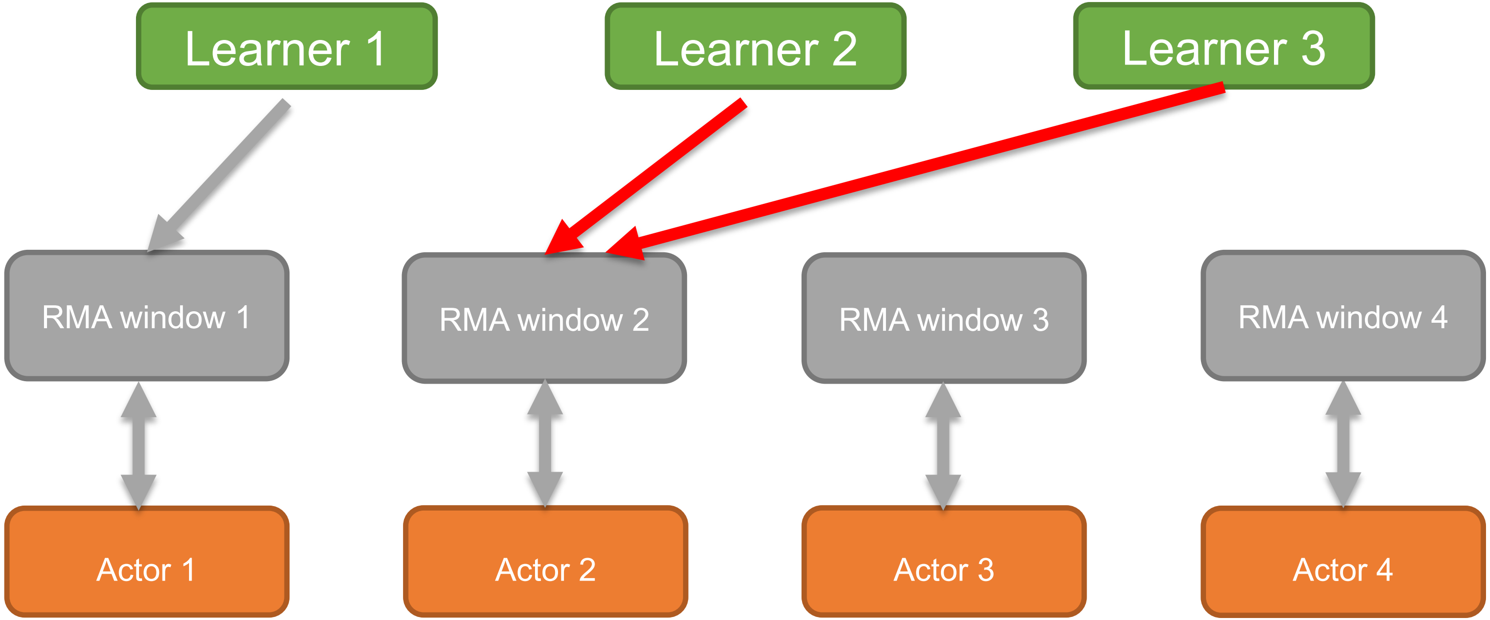
Current approach: Each learner randomly selects one of the actor's RMA window



Multi-learner RMA Workflow: Window-Selection Policy

In Multi-learner RMA workflow, learner gets the training data from the actor's RMA window

Limitation: Multiple learners access same actor window

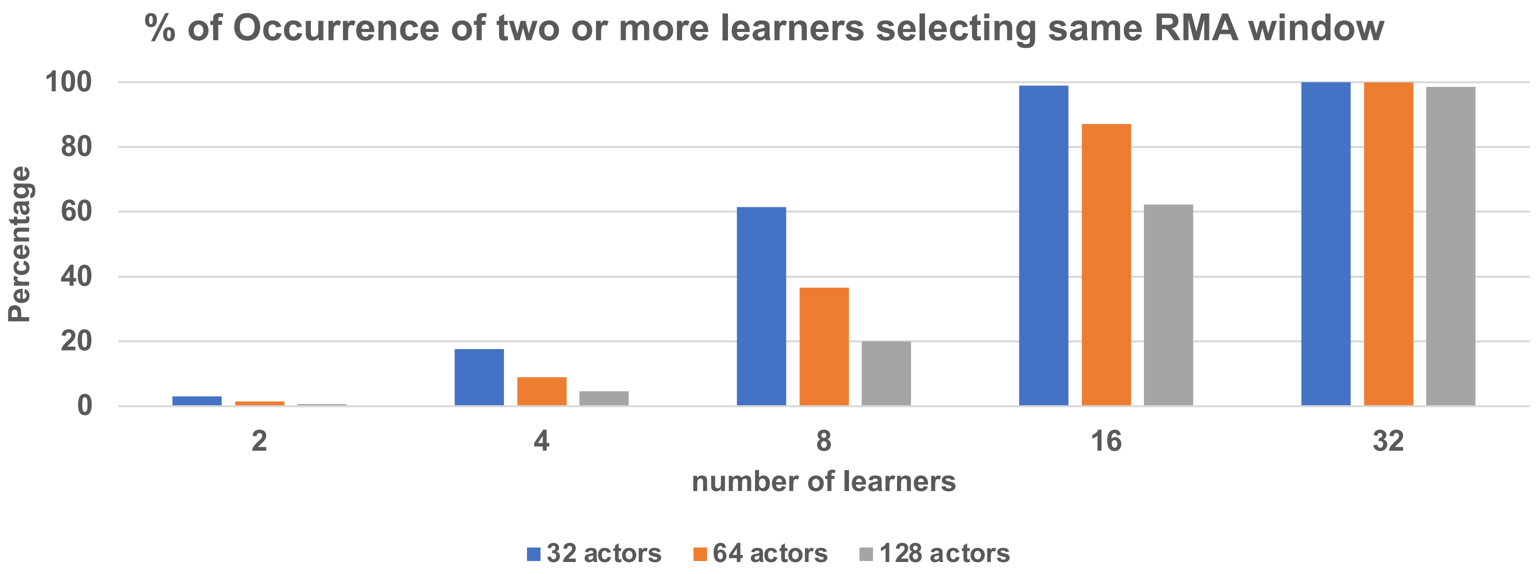


Multi-learner RMA Workflow: Window-Selection Policy

Performed simulations to observe the frequency of such behaviour

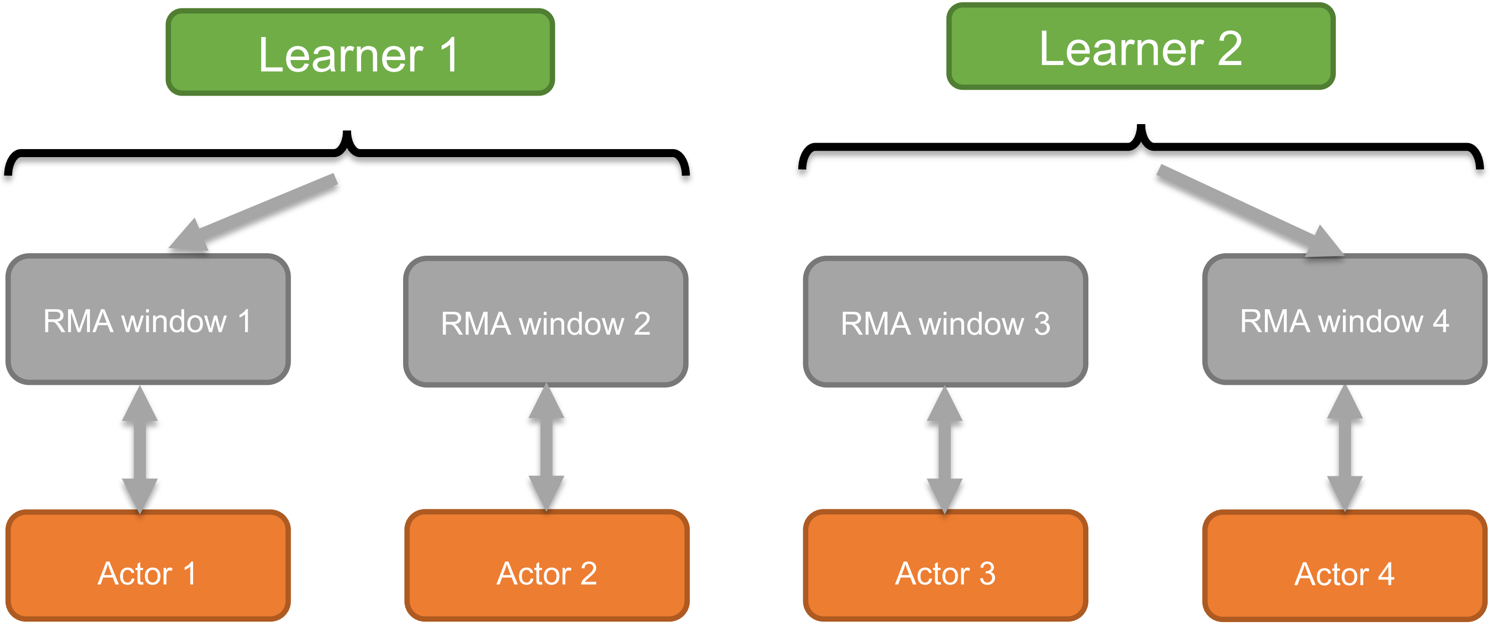
Observations:

- Significant occurrence when # of learners are at least 25% of the total actors



Multi-learner RMA Workflow: Window-Selection Policy

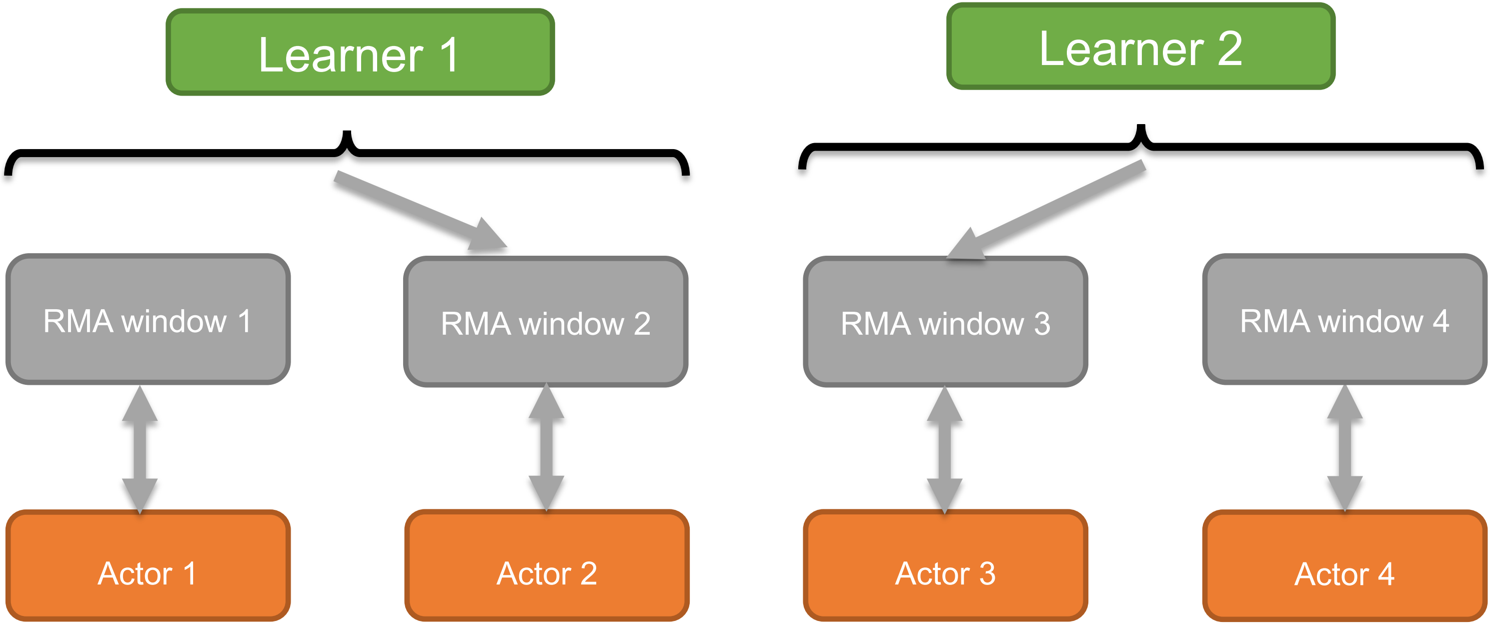
- **Proposed approach:**
  - Allocate a set of actor RMA windows to each learner



Los Alamos NATIONAL LABORATORY

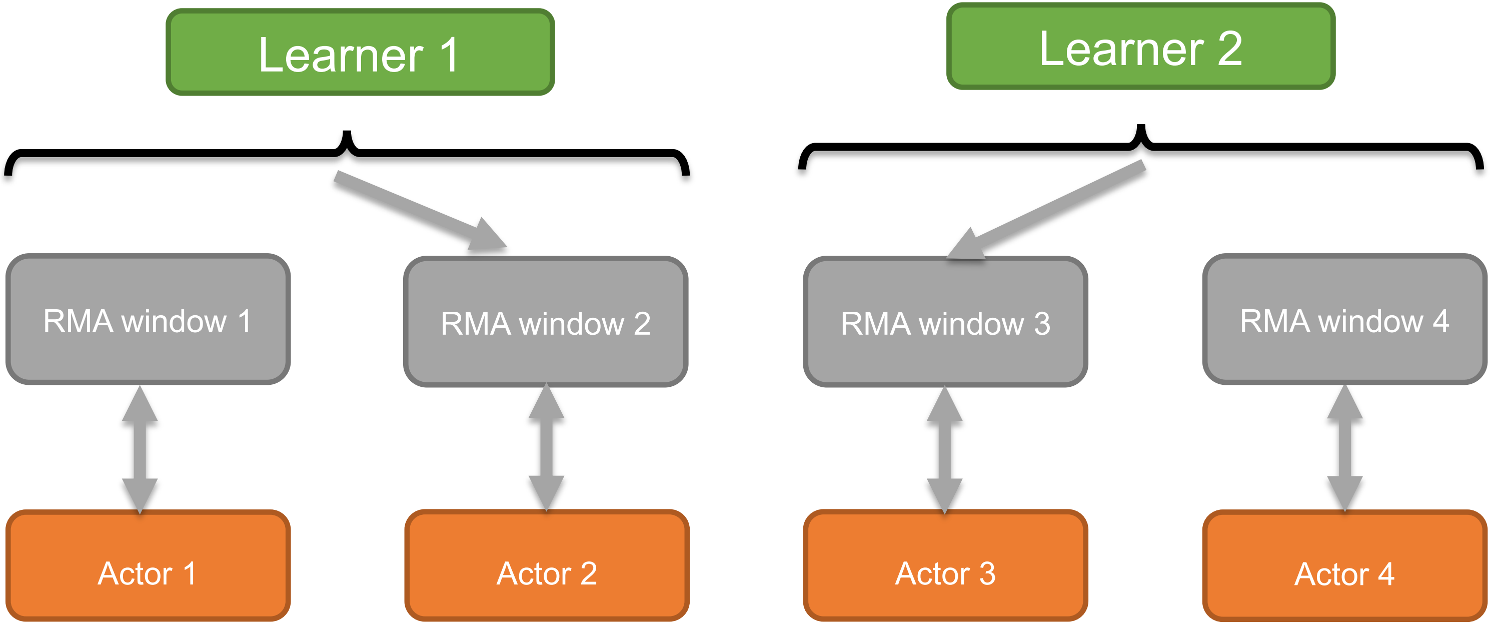
Multi-learner RMA Workflow: Window-Selection Policy

- **Proposed approach:**
  - Allocate a set of actor RMA windows to each learner



Multi-learner RMA Workflow: Window-Selection Policy

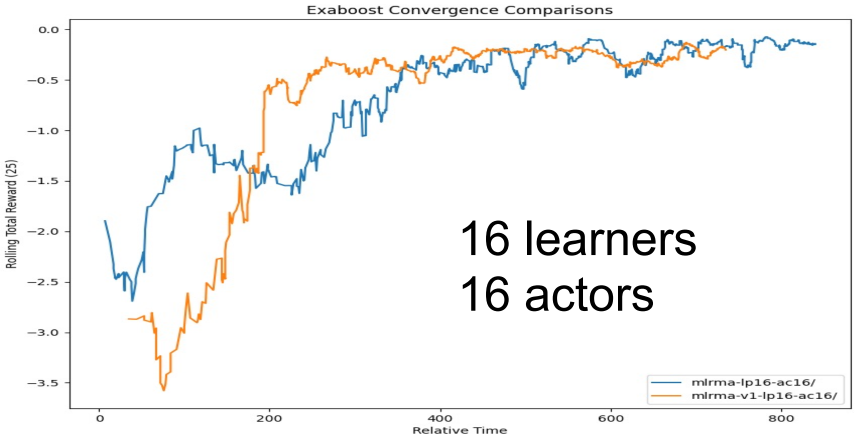
- **Advantages:**
  - Guarantees no learner reads from the same actor's RMA window

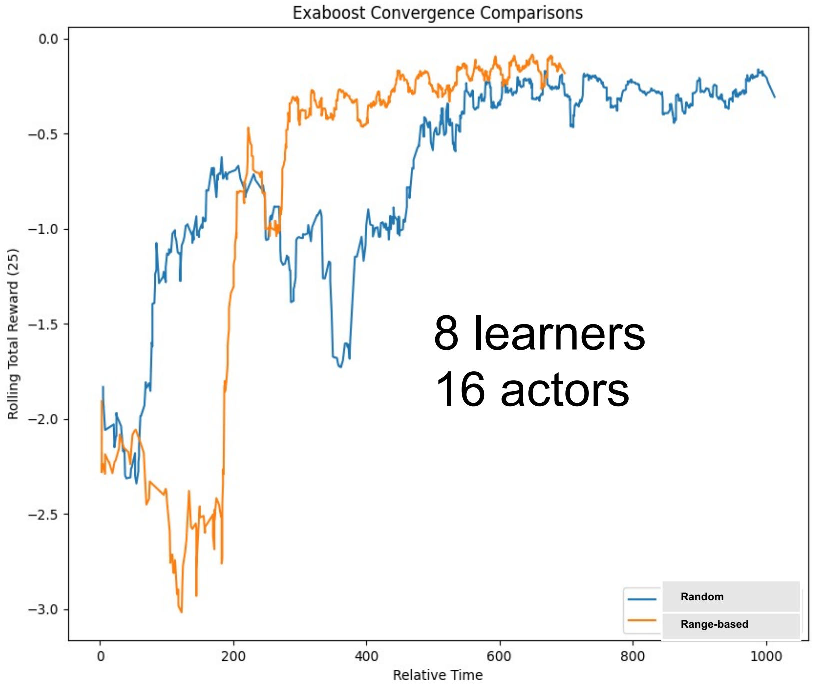


Multi-learner RMA Window Selection Policy: Results

- **Experimental Setup**:
  - Environment: ExaBooster
  - Episode count: 1000 Step count: 200
  - Action type: variable

- **Observations:**
  - Faster convergence
  - Improvement in convergence is due to non redundant training data during distributed learning

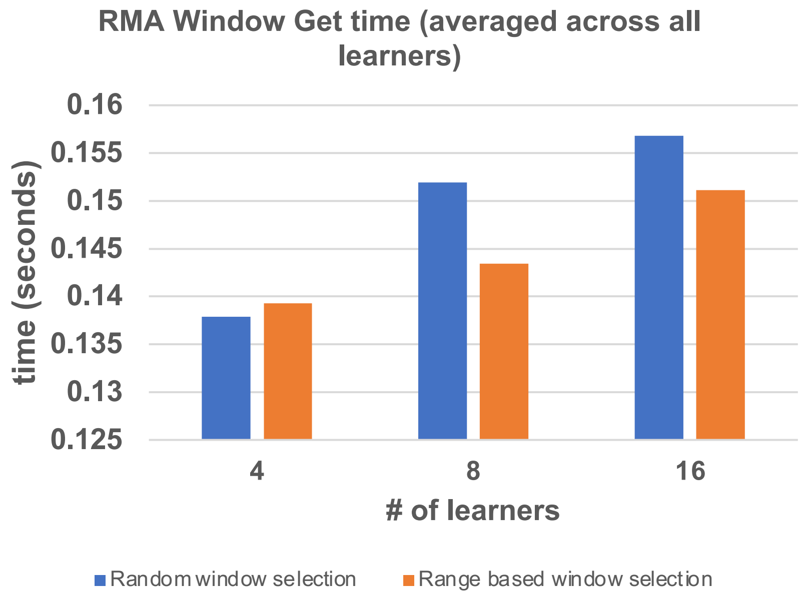


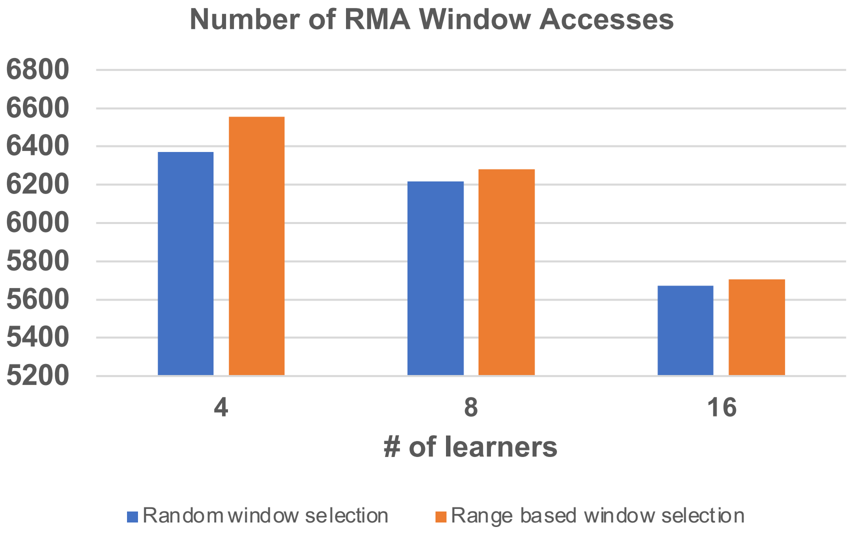


Multi-learner RMA Window Selection Policy: Results

- **Experimental Setup**:
  - Environment: ExaBooster
  - Episode count: 1000 Step count: 200
  - Action type: fixed

- **Observations:**
  - **Faster** convergence
  - (Not significant) **reduction** in access time.



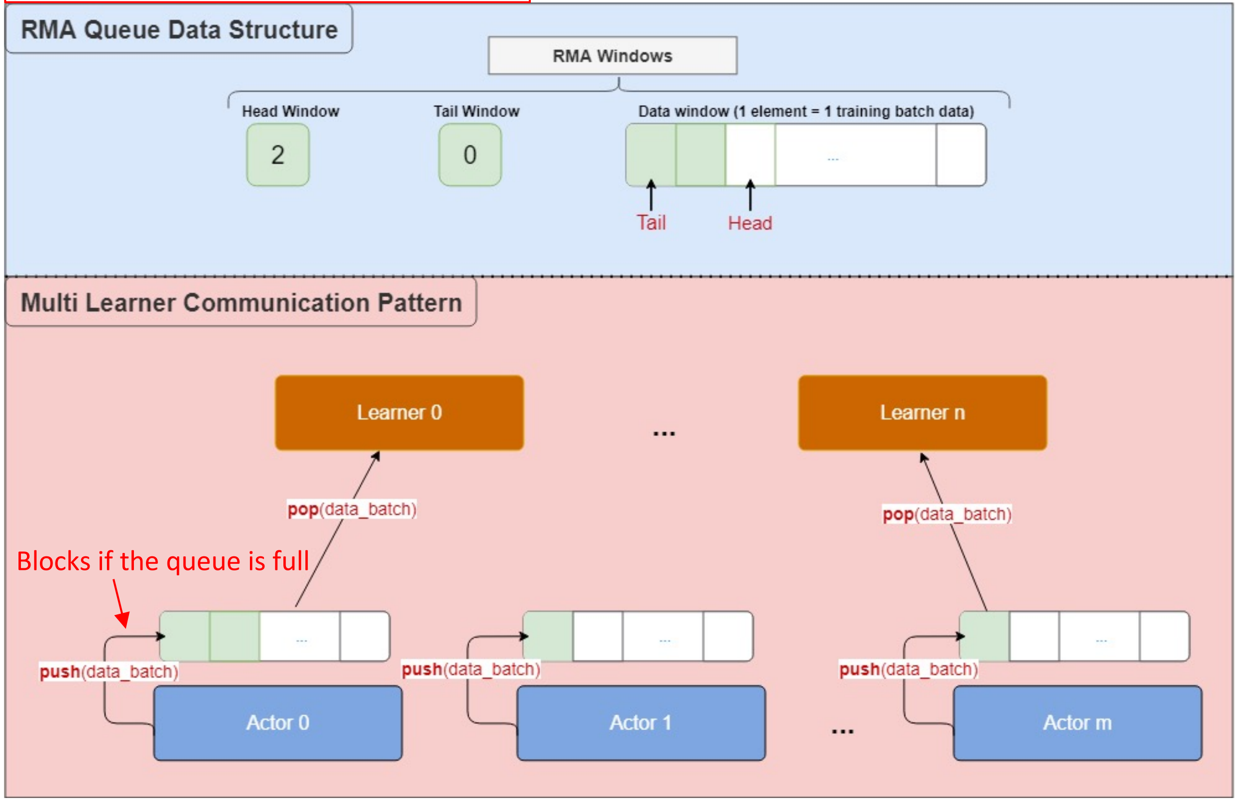


Los Alamos NATIONAL LABORATORY

***Tests run on Darwin (scaling partition): Intel Broadwell (36 cores) + IB ConnectX-4**

Multi-learner RMA Queue Asynchronous workflow

Data structure implemented by the EXARL team



Current approach

- Single learner.
- Communication pattern based on blocking MPI P2P routines.

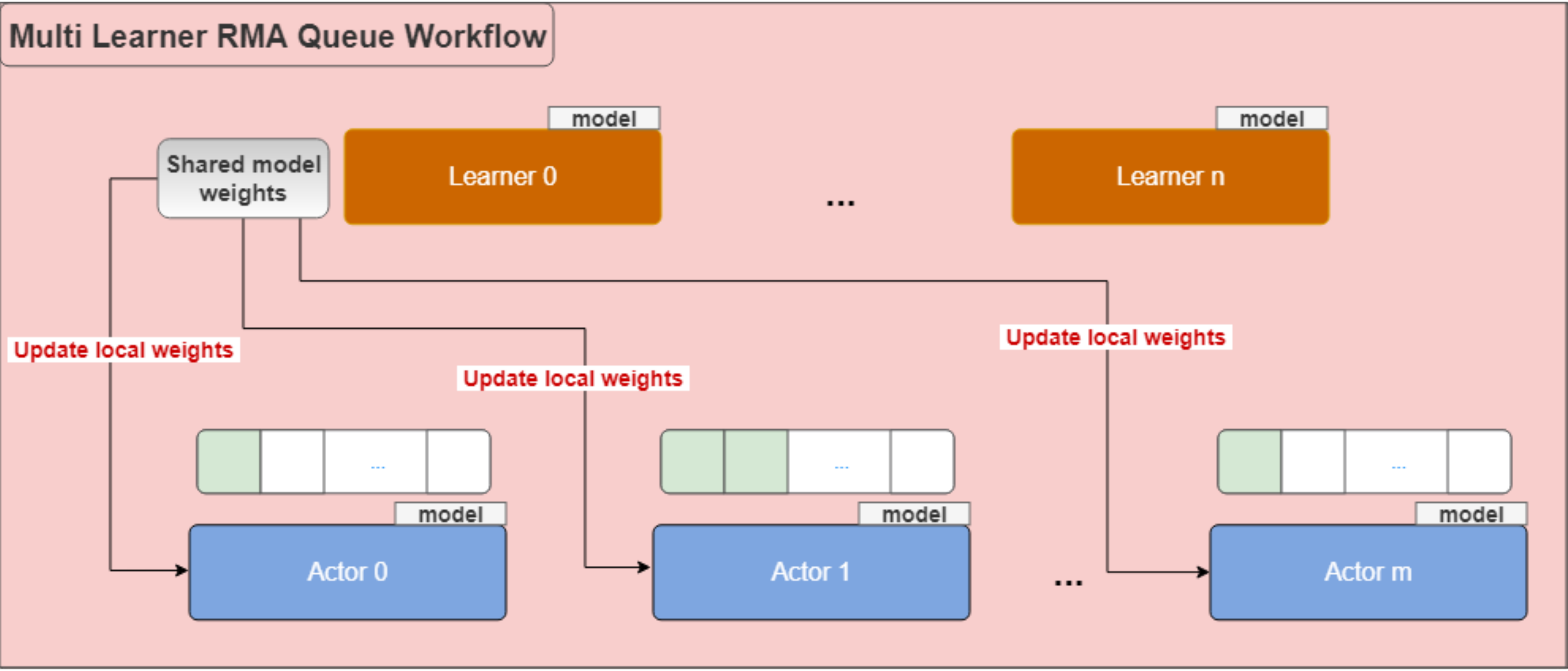
RMA Queue approach

- Use the queue data structure from the current EXARL RMA workflow.
- Multi-learner **communication pattern**:
    - Actors interact with the environment continuously and push batched training data to their local queue (blocks if the queue is full).
    - Each group of actors is assigned to a specific learner that pops training data randomly from its queue.

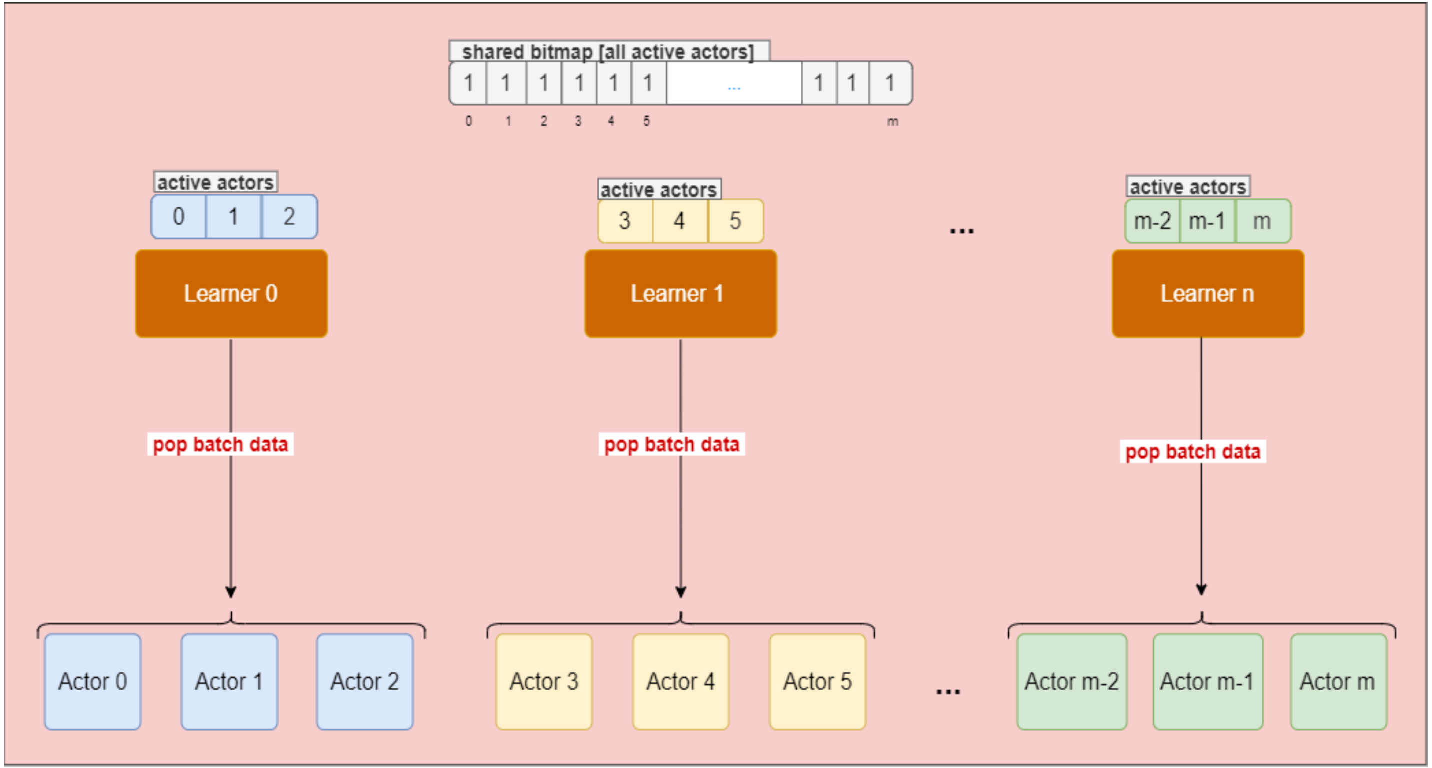
Advantages

- The actors and the learners are **decoupled**.
    - There is no active synchronization need.
- **Multi-learner** approach.

Multi-learner RMA Queue Asynchronous workflow



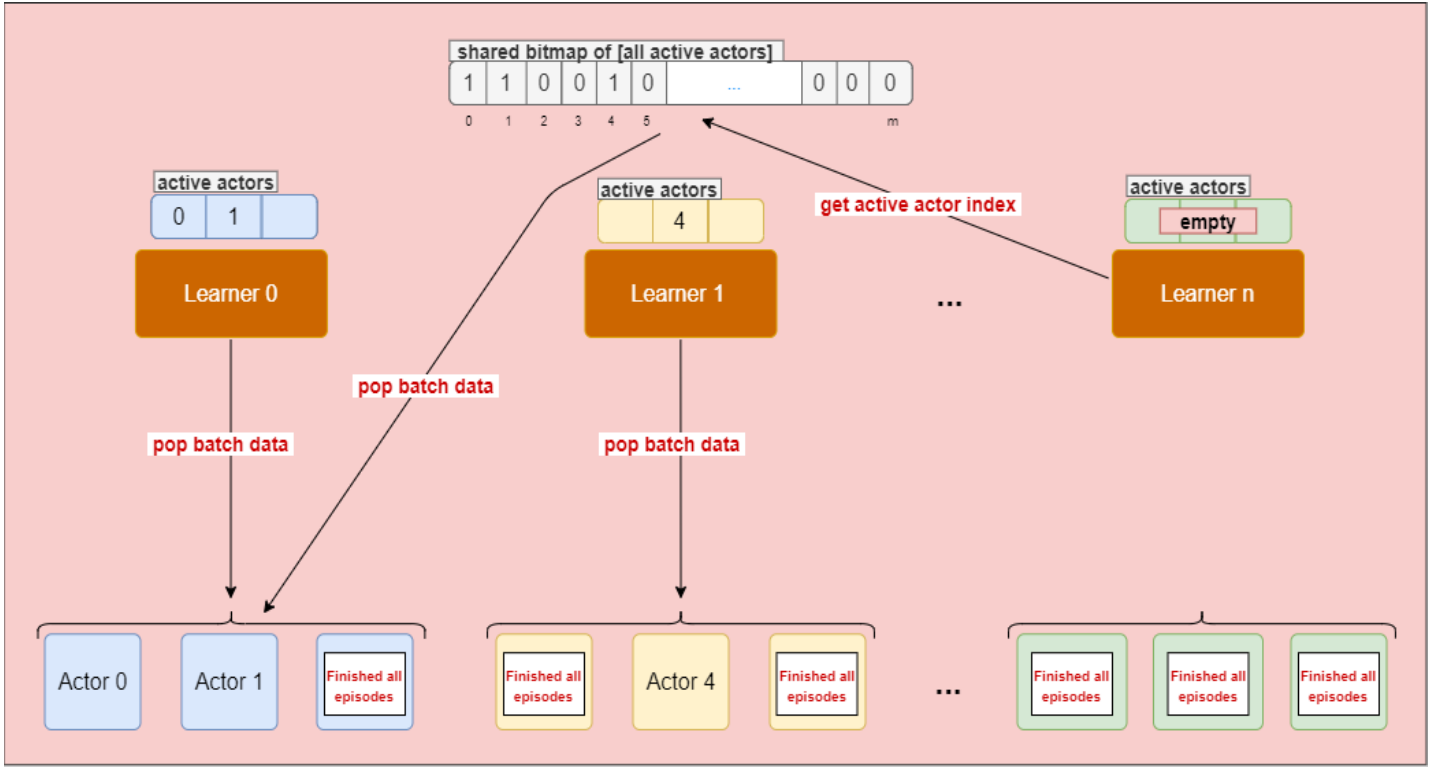
RMA Queue Asynchronous workflow – Implementation details



RMA Queue approach

- Each group of actors is assigned to a specific learner → allows to limit the number of simultaneous accesses to the same queue.

RMA Queue Asynchronous workflow – Implementation details

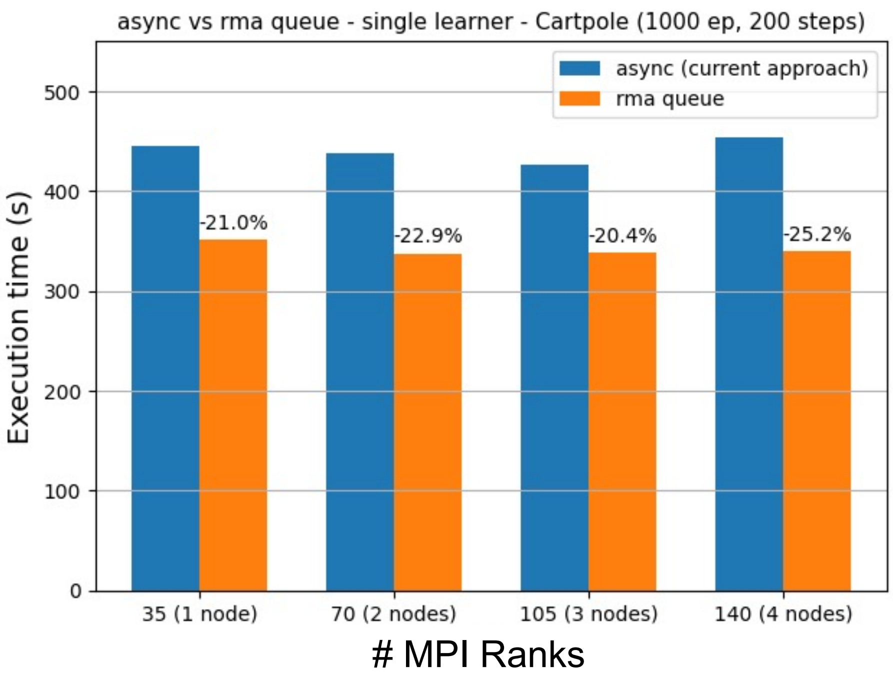


<u>RMA Queue approach</u>

- Learners that exhaust all ‘active’ actors assist other learners in fetching batch data.
- The “shared bitmap array” indicates which actors are active. This prevents getting data from an empty queue.

Los Alamos NATIONAL LABORATORY

Multi-learner RMA Queue Asynchronous workflow – results



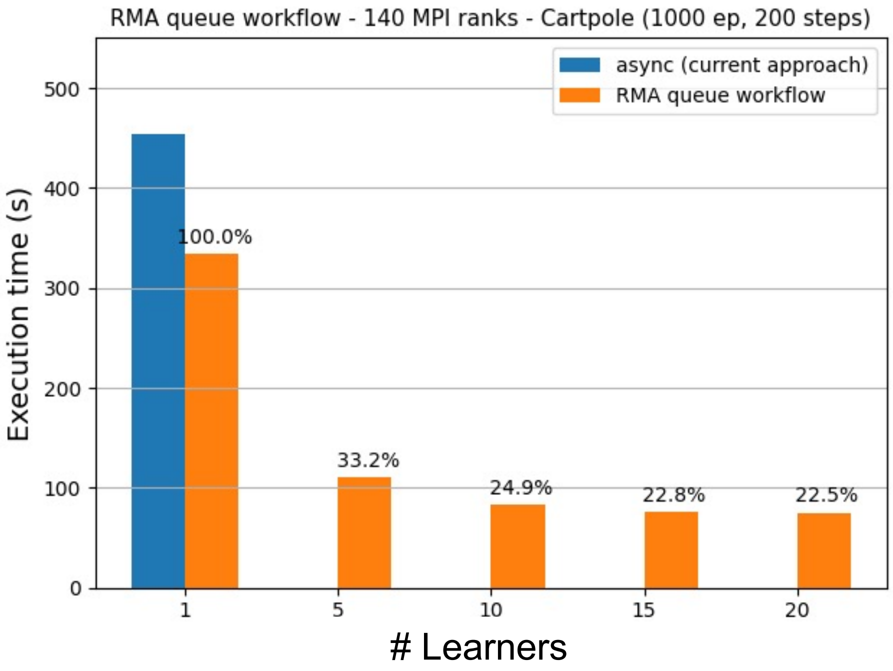
Single learner

- Achieved **25%** performance improvement (on 4 nodes) compared to current asynchronous workflow.
- Limited Scalability: adding more actors didn't decrease the execution time → obvious need to increase concurrency through adding learners.

Limited Scalability → need to increase the number of learners

***Tests run on Darwin (scaling partition): Intel Broadwell (36 cores) + IB ConnectX-4**

Multi-learner RMA Queue Asynchronous workflow – results

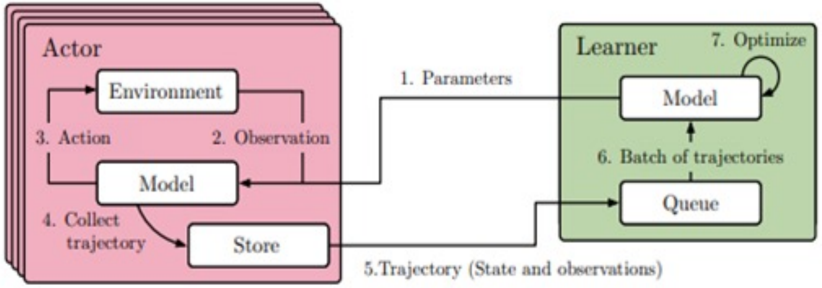


Multi-learner

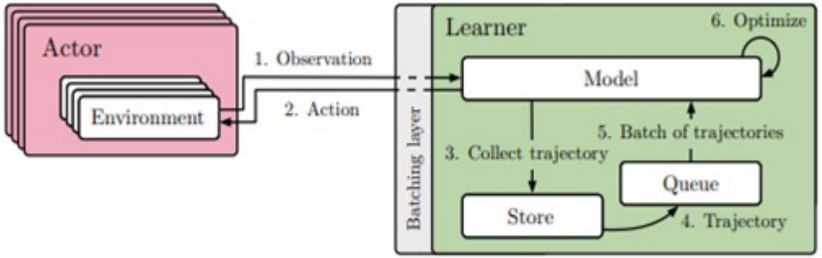
- Achieved **77%** performance improvements (using **20 learners**) compared to the single learner version.
- Good performance improvements for the **same amount of hardware resources** (140 processes).

*Tests run on Darwin (scaling partition): Intel Broadwell (36 cores) + IB ConnectX-4

SEED Architecture – moving the inference part to the learner



(a) IMPALA architecture (distributed version)



(b) SEED architecture

SEED RL : http://arxiv.org/abs/1910.06591

SEED Advantages

- Using GPUs for neural network inference can result in execution time performance improvements for larger models
- As there is only one copy of the model, there is no issue of copies going out of sync
- Low bandwidth requirements relative to model parameters

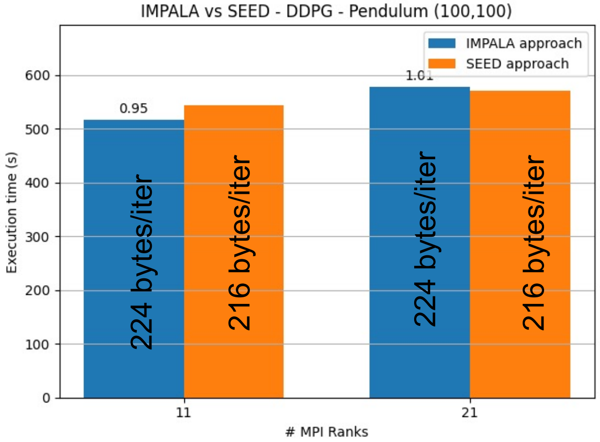
Current approach	SEED approach
Learner sends model weights	Learner sends the action to take
Actors send an entire training batch	Actors send a single observation

SEED Disadvantages

Can result in significant execution time increase for certain agents as the "generate_data()" function is called on the learner.

Example : DQN agent

SEED Results



***Tests run on Darwin (scaling partition): Intel Broadwell (36 cores) + IB ConnectX-4**

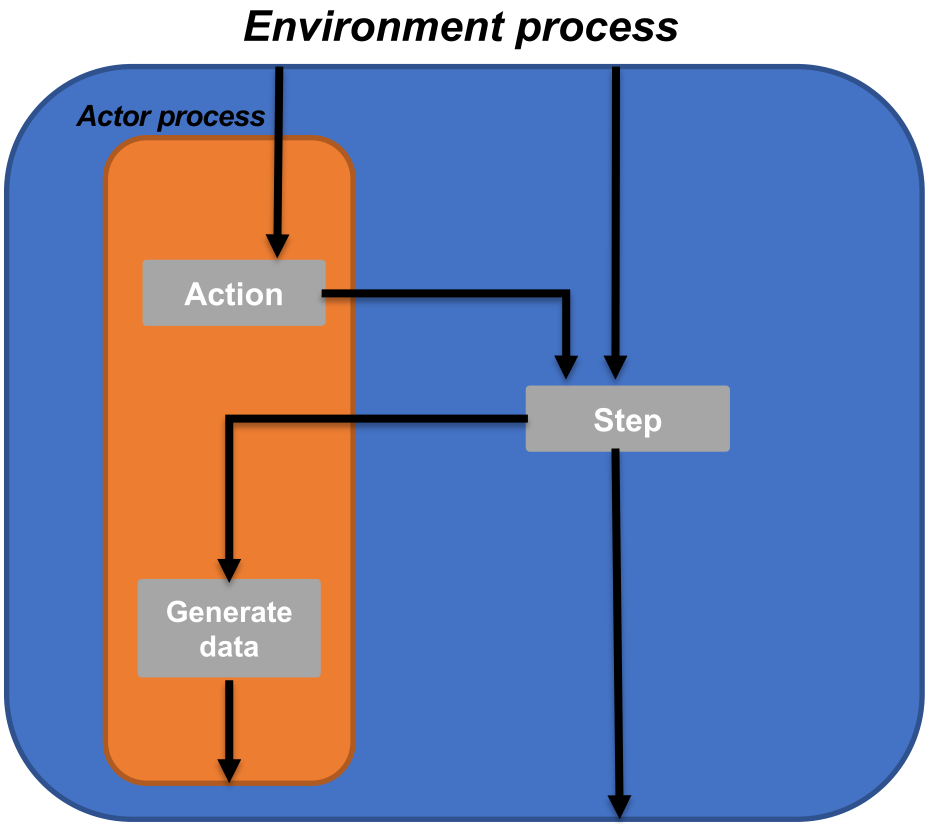
Accelerating DQN Data-Generation Pipeline

- Calculating Bellman optimality equation on each experience is expensive
  - *90%* of computation time
- **Current approach**: Actor generates the training data
- **Optimization**: Offload data-generation on remaining environment processes
- **Assumption**: actor and environment does not execute simultaneously

Sampled experiences

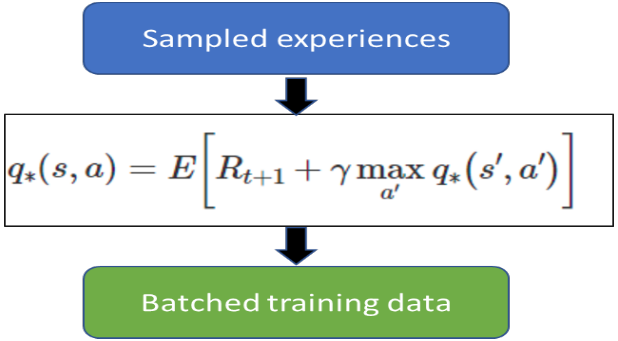
$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

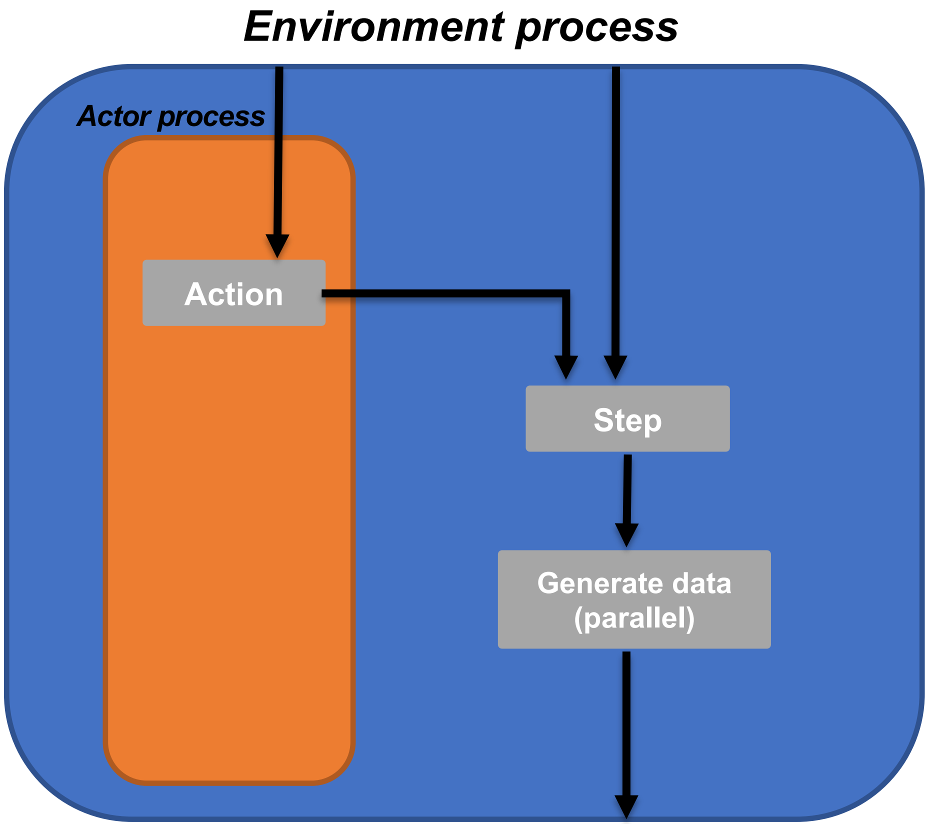
Batched training data



Accelerating DQN Data-Generation Pipeline

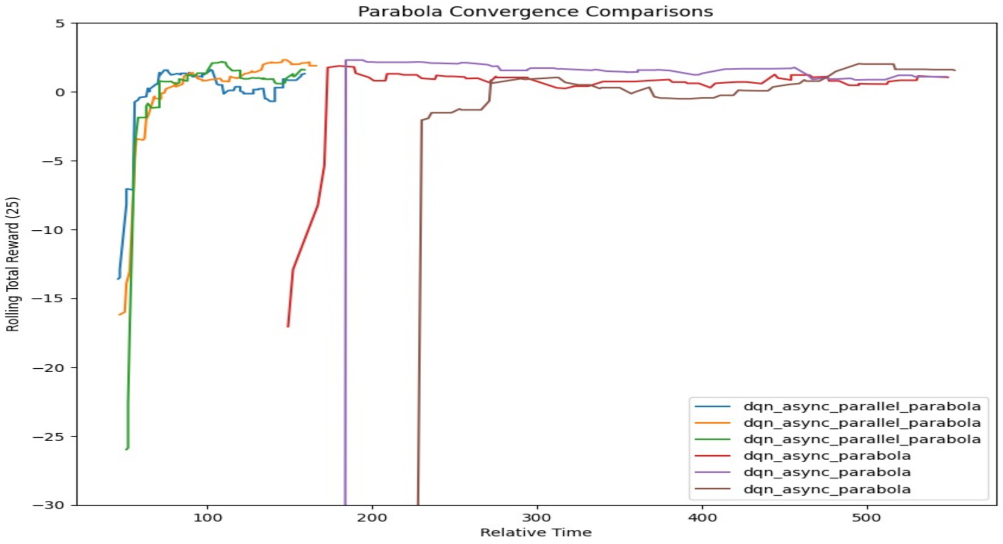
- Calculating Bellman optimality equation on each experience is expensive
  - *90%* of computation time
- **Current approach**: Actor generates the training data
- **Optimization**: Offload data-generation on remaining environment processes
- **Assumption**: actor and environment does not execute simultaneously

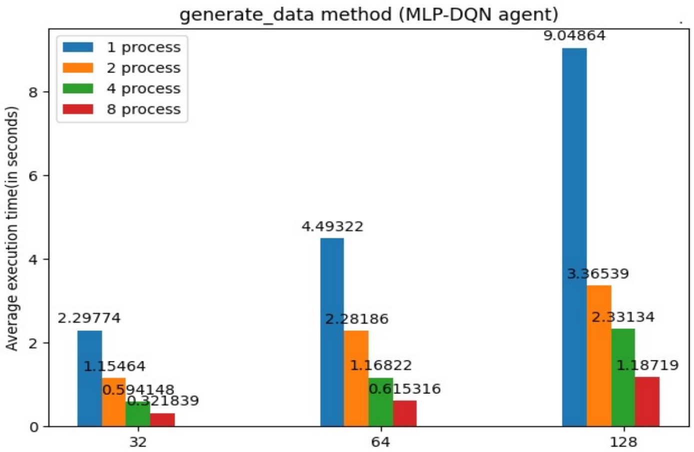


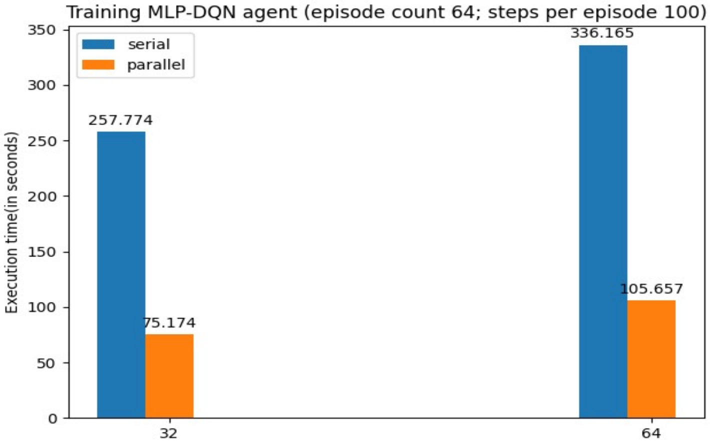


Accelerating DQN Data-Generation Pipeline

- **Results**:
  - Average speedup of 3.30x upon scaling the workload to 4 processes
  - Faster convergence





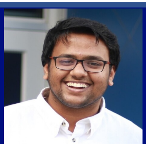


Overview

Main goal of the Co-Design Summer School 2021 is to provide **algorithmic improvements to EXARL framework**. This is in the form of:

<u>**Improving Performance**</u>

- Scaling asynchronous workflow to multiple learners
- Improve scalability/execution time of multi-learner workflows
- Accelerate Deep Q-Network (DQN) data generation pipeline




<u>**Adding Functionalities**</u>

- New agents: Advantage Actor Critic (A2C/A3C), Twin Delayed Deep Deterministic Policy Gradient (TD3)
- V-trace algorithm
- Priority Experience Replay


(Asynchronous) Advantage Actor Critic (A2C/A3C)

Current Available Agent:

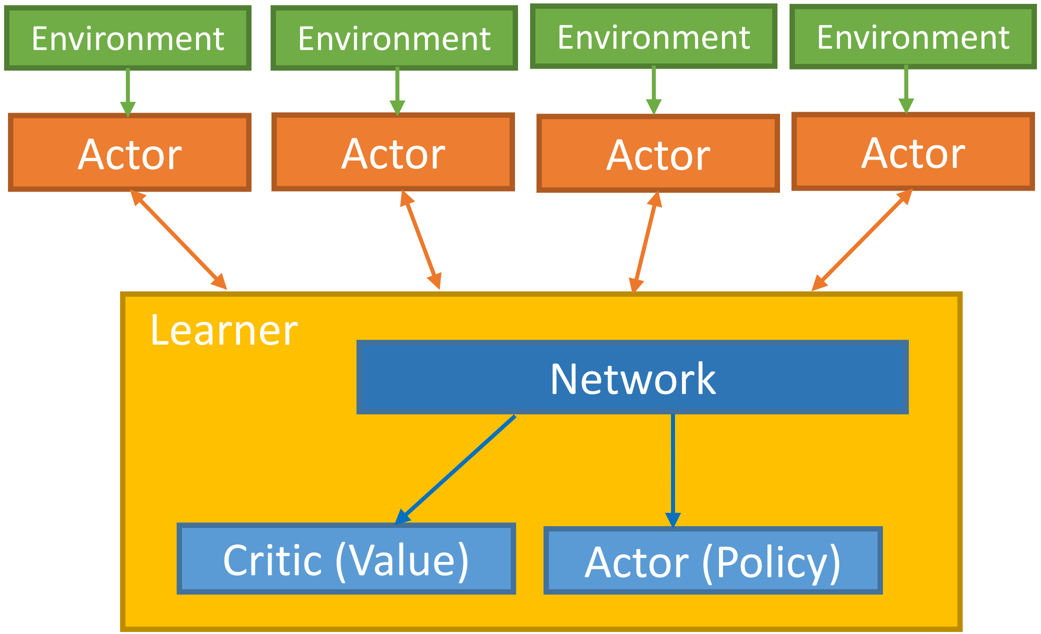
- Deep Q-Network (DQN)

Limitations:

- DQN often takes a long time to train because it uses old data from replay buffer
- Training time is also long because of calculation of Bellman Equation

Update:

- A2C: synchronous workflow
- A3C: asynchronous workflow
- Faster to train & with more diverse data because each worker has their own environment for generating trajectories
- Current implementation is for <u>discrete action space environments</u>, but can be formulated for continuous ones, as well.



(Asynchronous) Advantage Actor Critic (A2C/A3C)

Current Available Agent:

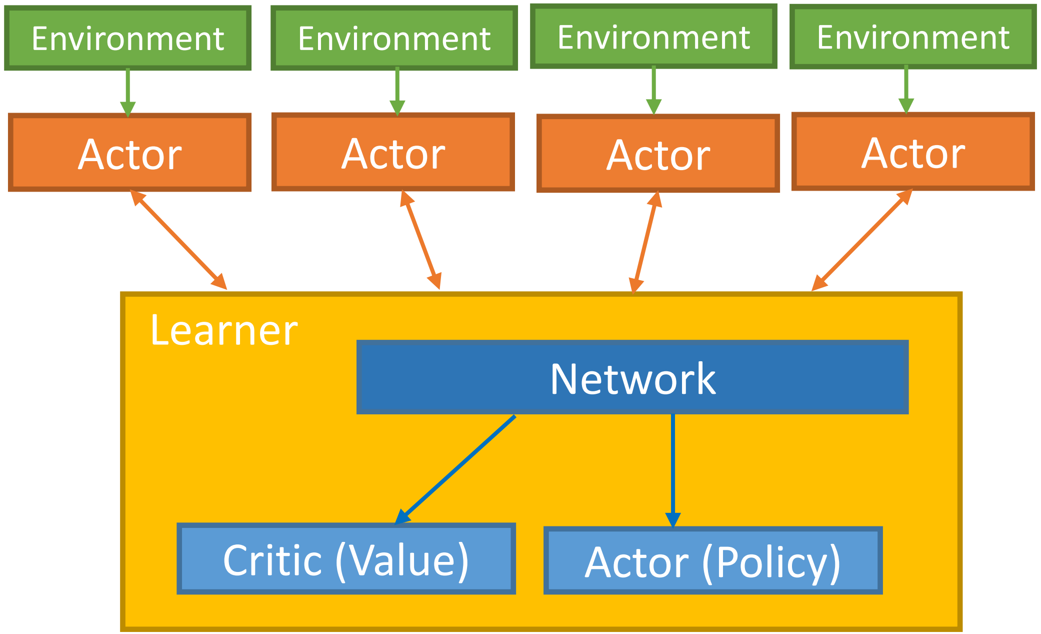
- Deep Q-Network (DQN)

Limitations:

- DQN often takes a long time to train because it uses old data from replay buffer
- Training time is also long because of calculation of Bellman Equation

Update:

- A2C: synchronous workflow
- A3C: asynchronous workflow
- Faster to train & with more diverse data because each worker has their own environment for generating trajectories
- Current implementation is for <u>discrete action space environments</u>, but can be formulated for continuous ones, as well.

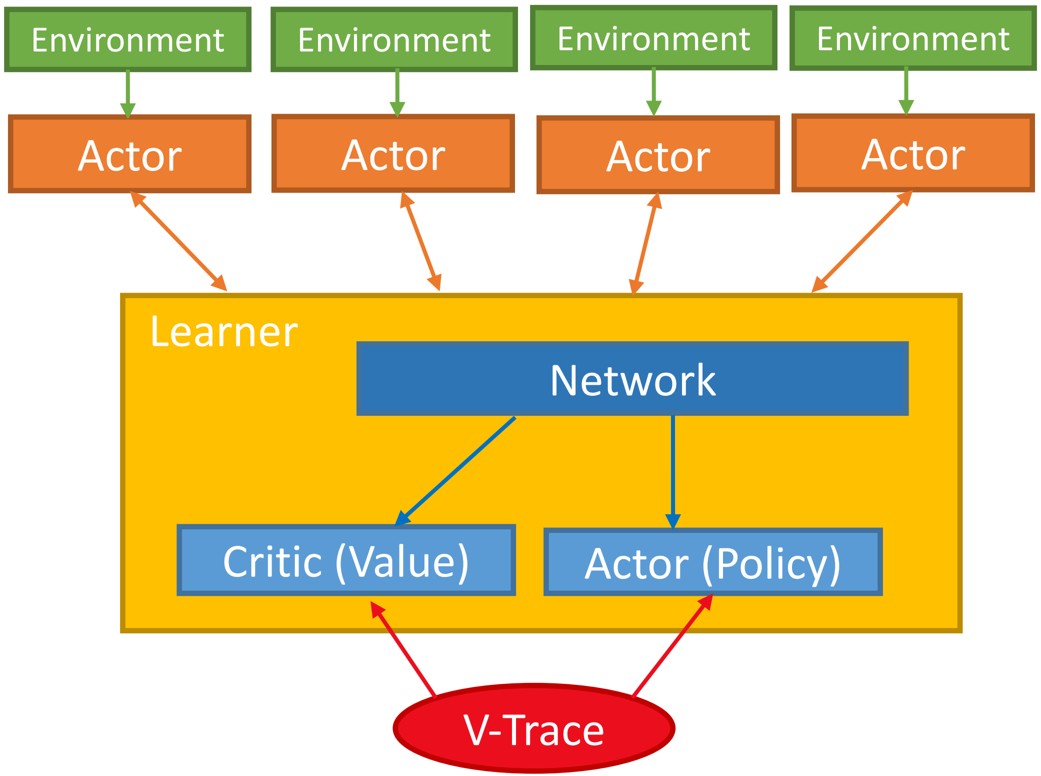


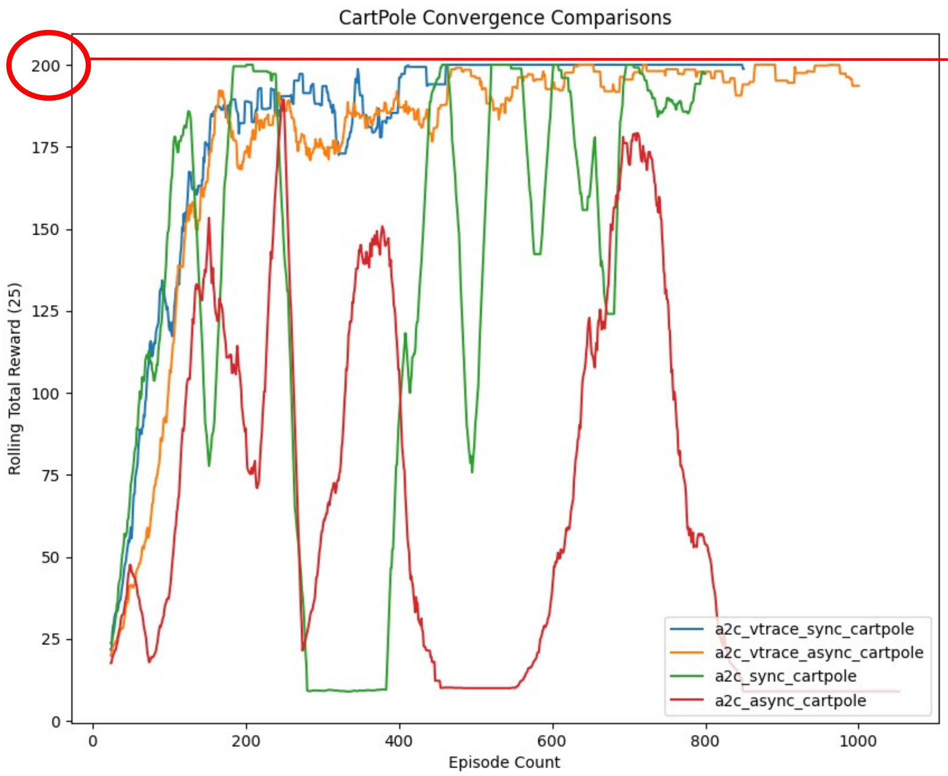
e.g. move left or right

e.g. move according to an applied torque

(Asynchronous) Advantage Actor Critic + V-Trace

- **On-policy**: the policy an actor acts with should be the same as the policy a learner learns with.
- In the **EXARL framework**, we can't always guarantee that they will have the same policy.
- To correct for that, we add an algorithm called **"v-trace"** to the loss functions.
- This correction assumes that the ratio between the two policies is always equal to one, therefore its addition forces this condition and **we obtain the required on-policy behavior.**

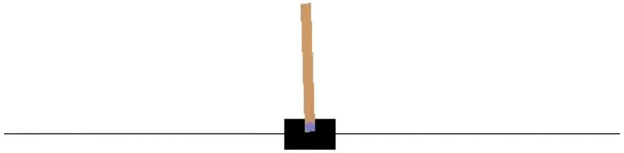




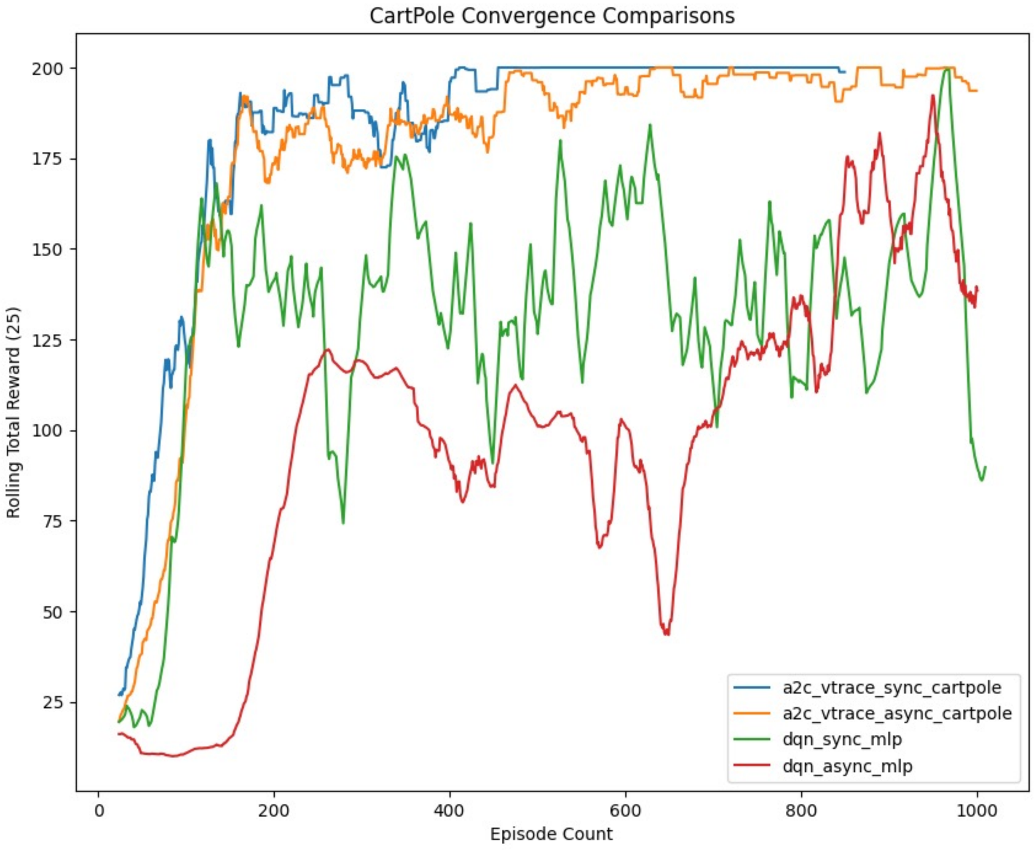
Max. reward for CartPole is 200.

The agent successfully balances the pole on the cart for 200 consecutive time steps (call this **convergence**).

A2C/A3C *with* v-trace reaches convergence, while A2C/A3C *without* v-trace does not.

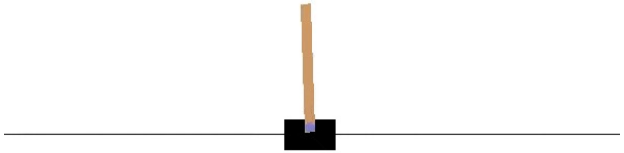


www.martinholub.com



A2C/A3C converge to expected value of 200 (CartPole environment), however DQN does not.

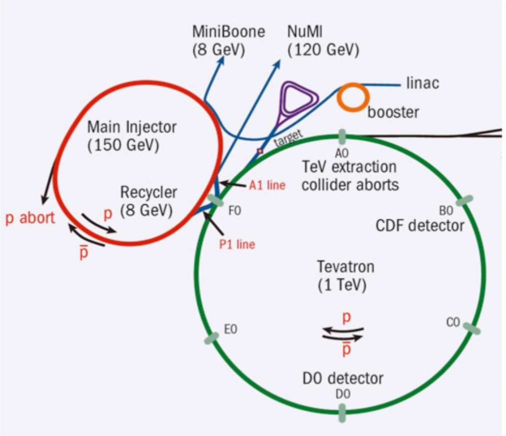
Results show DQN with Multi-Layer Perceptron (MLP) network, however results are similar for DQN with Long Short-Term Memory (LSTM) network.

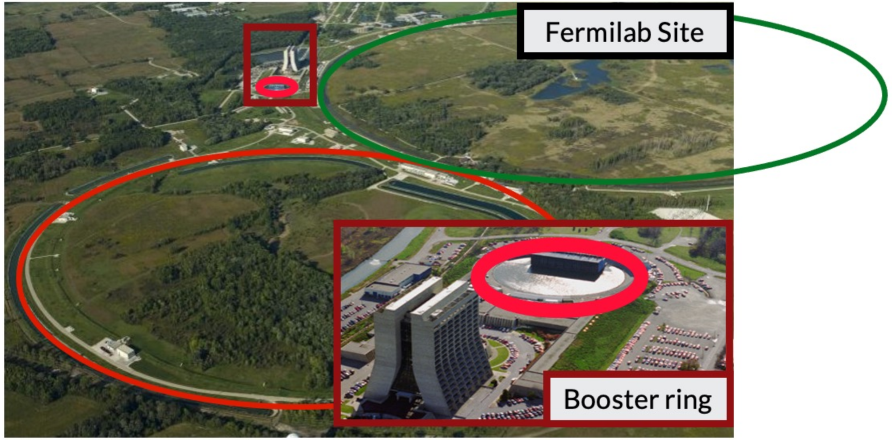


www.martinholub.com

ExaBooster Environment

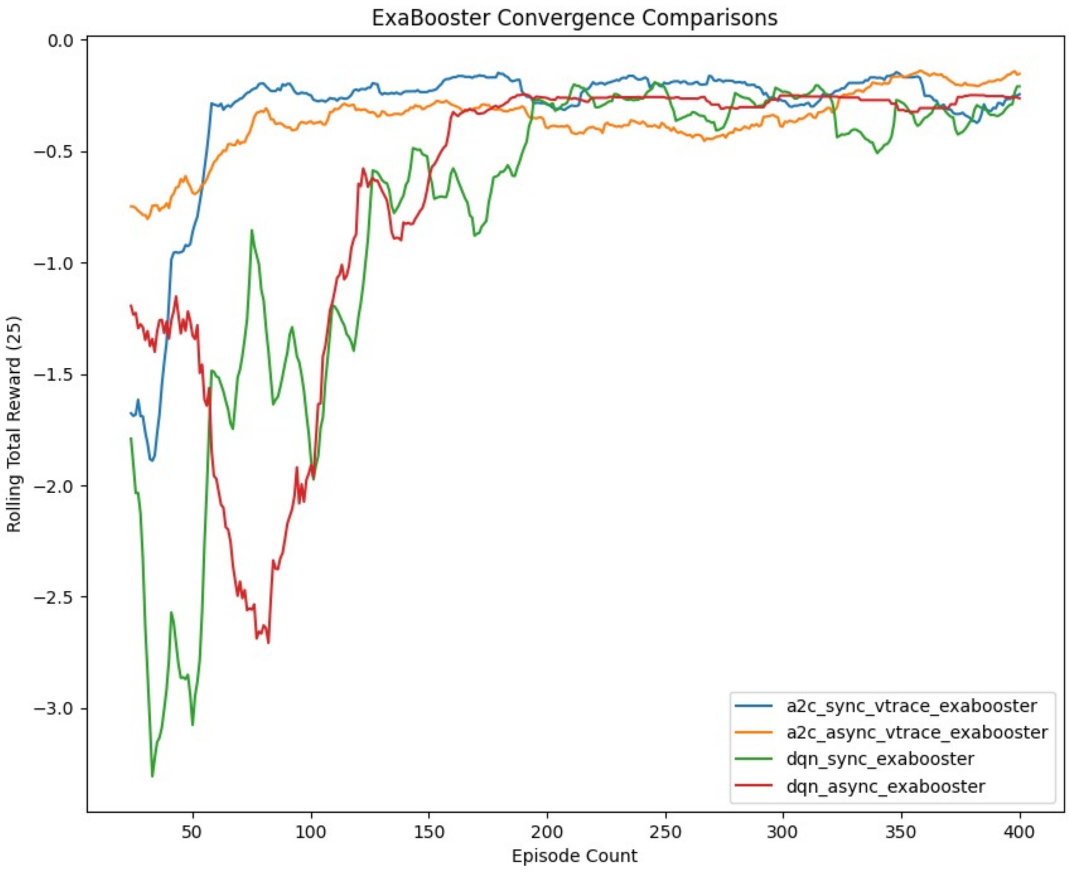
FNAL Accelerator Complex:





Courtesy: Christian Herwig

- Control problem for FNAL particle accelerator at FermiLab.
- Reinforcement learning is used to control particle beam quality (ie. reduce beam losses) in real time.
- Keeps the beam field from spreading (thus degrading the beam quality) by regulating the magnetic current of the booster.
- Original work developed by PNNL, FNAL, University of California San Diego, Columbia University



Convergence means: magnetic current is within some tolerance of an optimal value, which prevents too much spread in the beam field

A2C/A3C converges slightly faster than DQN (with LSTM network)

Added Agents

Current Available Agent for Continuous Action Space:

- Deep Deterministic Policy Gradient (DDPG)

Limitations:

- It is frequently brittle to hyperparameters and other kinds of tuning
- The learned Q-function begins to overestimate Q-values which leads to policy breaking

Additions:

- Twin Delay Deep Deterministic policy gradient agent
- Prioritized Replay Buffer with Sumtree

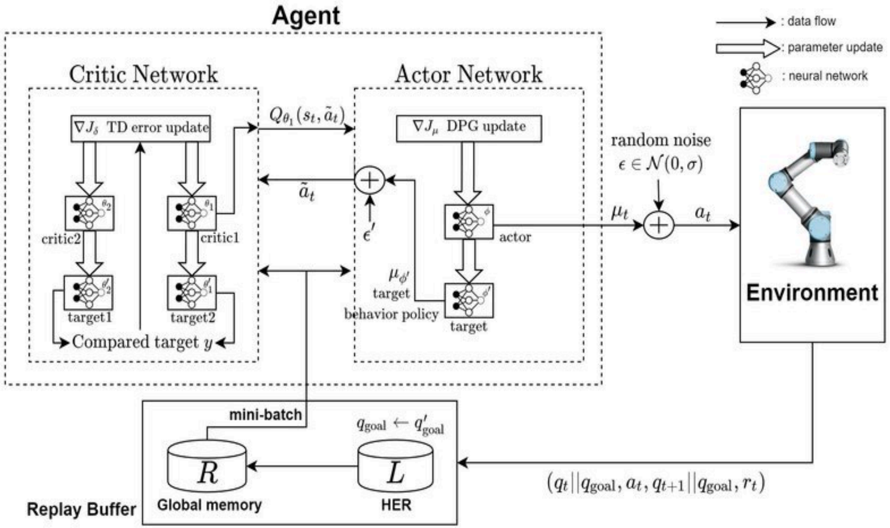
Twin Delay Deep Deterministic Policy Gradient (TD3)

Off-policy Agent: the policy an actor acts is independent on the policy a learner learns.

Twin Delay Deep Deterministic Policy Gradient Agent (TD3):

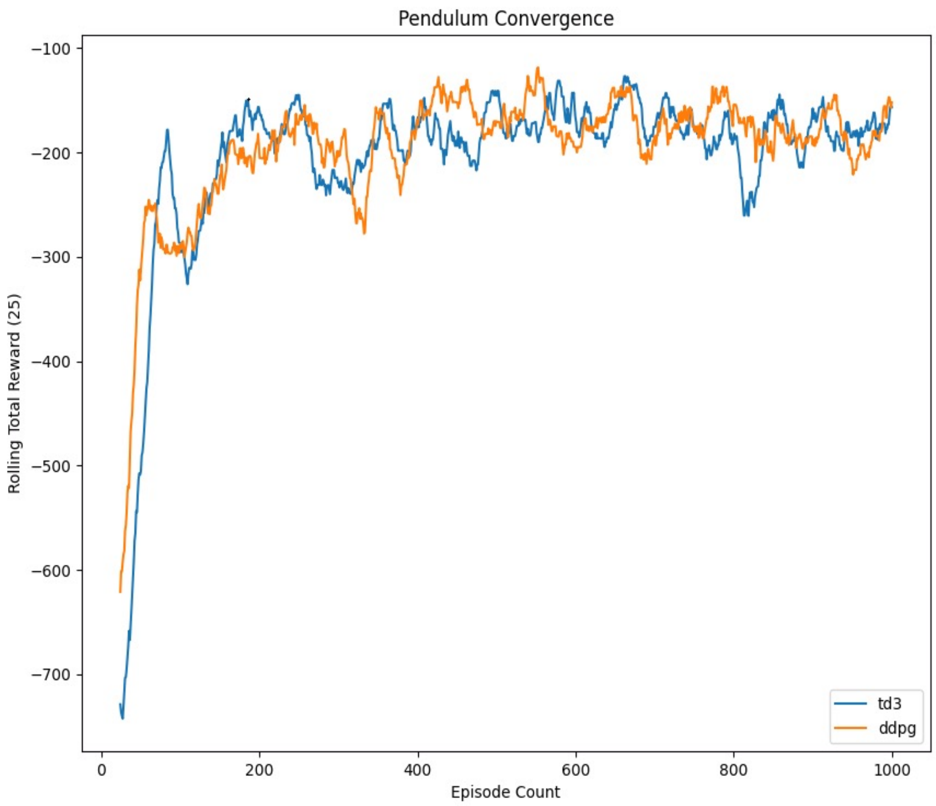
- Address the overestimate issue by using 3 tricks
  - Clicked Double Q-learning: Learns two Q-functions and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
  - Delayed Policy agent: Updates the policy and target networks less frequently than the Q-function
  - Target policy smoothing: TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along the changes in action

TD3 Architecture

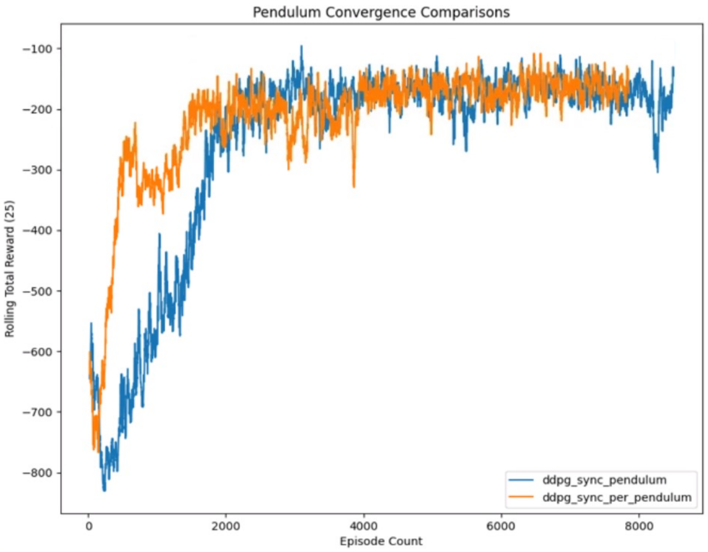


https://www.researchgate.net/figure/Structure-of-TD3-Twin-Delayed-Deep-Deterministic-Policy-Gradient-with-RAMDP_fig2_338605159

TD3 vs DDPG (Synchronous workflow)



Effect of Replay Buffer

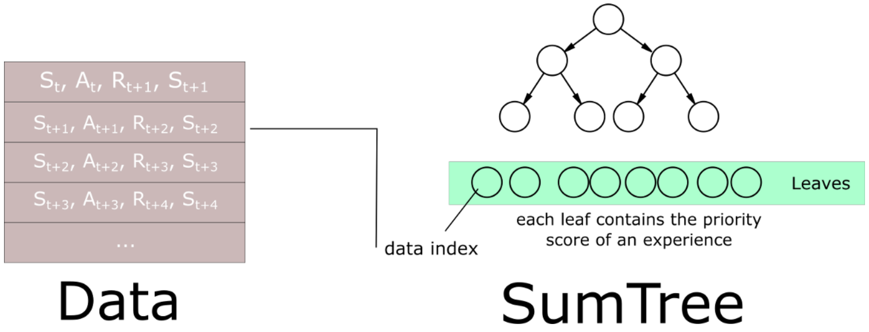


Uniform Sampling Replay Buffer

- Each transition sample in the minibatch is sampled uniformly from pool of stored experiences
- **Limitation**: When treating all samples the same, we are not using the fact that we can learn more from some experience

Prioritized Experience Replay Buffer

- Ranking of the experiences using the temporal-difference (TD) error (difference between the Q function and its target).
- Ranking of experiences by TD-error was done by storing the priority to experience mapping in a sum-tree.
- To avoid overfitting of our agent we update our policy network with important sampling weights.
- Sum-tree takes O(log n) for updating the tree and 0(1) to get the highest priority.



https://pylessons.com/CartPole-PER/

Summary/Conclusions: Performance Improvements

- We demonstrated improved scalability performance using efficient **RMA communication patterns**.
- Here we found that the total execution time decreased by 77% while using 20 learners and 120 actors on 4 nodes.
- We also created a **multi-learner asynchronous workflow**.
- Here we found there was a 43.4% increase in training throughput with 8 learners (actors = 8), training time reduced by 31% (actors = 16; learners = 8)
- We improved upon the existing framework by **accelerating the data generation pipeline** for the DQN agent for faster convergence.
- Here we found an average speedup of 3.30x when scaling the workload to 4 processes.

Summary/Conclusions: Adding Functionalities

- We expanded the capability of EXARL by including additional agents like (Asynchronized) Advantage Actor Critic **(A2C/A3C)** and Twin Delayed Deep Deterministic Policy Gradient **(TD3)**
- We also explored algorithmic improvements such as **v-trace** and **Prioritized Experience Replay**
- Here we found that **A2C/A3C** performed best with **v-trace** and outperformed Deep Q-Network (DQN) on both the CartPole game and the ExaBooster scientific environment.
- We also found that **TD3** performed as good as the existing Deep Deterministic Policy Gradient (DDPG) agent
- We saw that adding **Prioritized Experience Replay** to DDPG accelerated convergence.

Acknowledgements

The Co-Design Summer School mentors:

- Vinay Ramakrishnaiah
- Robert Pavel
- Julien Loiseau
- Hyun Lim
- Jamal Mohd-Yusof
- Andrew Reisner
- Karen Tsai

CCS-7, especially Christoph Junghans, Erika Maestas

ECP, especially Christine Sweeney

Parallel Computing Summer Research Internship, especially Bob Robey

ExaLearn is funded by NNSA and the DOE Office of Science

U.S. DEPARTMENT OF ENERGY

Office of Science

National Nuclear Security Administration

Center for Nonlinear Studies

Information Science & Technology Institute

EXASCALE COMPUTING PROJECT

Questions?